


零点起飞学编程









- 本书实例源文件
- **16小时**配套教学视频
- **13小时**进阶视频讲座
- **12个**典型模块源文件
- **6个**拓展项目案例源文件





零点起飞学PHP

29小时高清多媒体教学视频

张少卓　等编著

- **循序渐进**：基础→进阶→实战
- **科学编排**：基本语法→典型实例→编程练习→项目实战
- **学练结合**：**331个**实例、**2个**项目案例、**53个**练习题
- **视频讲解**：提供配套**多媒体教学视频**
- **有问必答**：提供**QQ群**、**E-mail**和**论坛**答疑服务

清华大学出版社

零点起飞学编程

零点起飞学 PHP

张少卓　等编著

清华大学出版社
北　京

内 容 简 介

本书结合大量实例，由浅入深、循序渐进地介绍了 PHP 开发技术。书中内容丰富，图文并茂，讲解时理论与实例相互渗透，力图以最直观的方式使读者学习各个知识点，是一本简单易懂，易学易用的书。**本书特意提供了典型习题及教学 PPT，以方便教学。另外，本书配有大量配套教学视频，以帮助读者更好地学习。这些视频和书中的实例源代码一起收录于本书的配书光盘中。**

本书共 14 章，分为 3 篇。第 1 篇介绍了 PHP 基础知识和规范，主要包括 PHP 环境的搭建、PHP 的数据类型与运算符、语言结构、函数、数组、面向对象及错误处理等；第 2 篇介绍了 PHP 开发进阶技术，主要包括字符串处理、文件系统操作、图像处理、数据库管理、Cookie 和 Session 等；第 3 篇为开发实战篇，主要介绍了面向实际应用的网站模版和常用模块的开发过程，以提升读者的实际开发水平。

本书适合 PHP 入门与提高人员阅读，也可作为大中专院校及职业院校 PHP 开发课程的教材。另外，本书也可供网站开发程序员和编程爱好者作为实际工作中的参考书籍。

图书在版编目（CIP）数据

零点起飞学 PHP / 张少卓等编著. —北京：清华大学出版社，2013.9
（零点起飞学编程）
ISBN 978-7-302-31951-1

Ⅰ. ①零…　Ⅱ. ①张…　Ⅲ. ①PHP 语言 – 程序设计　Ⅳ. ①TP312

中国版本图书馆 CIP 数据核字（2013）第 078115 号

责任编辑： 夏兆彦
封面设计： 欧振旭
责任校对： 徐俊伟
责任印制： 何　芊

出版发行： 清华大学出版社
网　　址：http://www.tup.com.cn, http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-62770175　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者： 北京世知印务有限公司
装 订 者： 北京市密云县京文制本装订厂
经　　销： 全国新华书店
开　　本： 185mm×260mm　　**印　　张：** 21.75　　**字　　数：** 546 千字
附光盘 1 张
版　　次： 2013 年 9 月第 1 版　　**印　　次：** 2013 年 9 月第 1 次印刷
印　　数： 1～3000
定　　价： 49.80 元

产品编号：051510-01

前　　言

随着互联网的发展，网站已经成为人们生活中重要的组成部分。人们通过网站提供的各项功能进行购物、求职、阅读……现在3G技术的推广，移动应用和Web应用也成为开发热点，与两者相关的技术都得到了充分应用。PHP作为服务器技术得到了广泛应用，由于其开源免费，一直是网站开发的三大主流技术之一。同时，由于PHP开发简便、灵活，运行效率高，也成为移动应用和Web应用的服务器端主流技术。

成也萧何，败也萧何。开源技术成就了PHP主流开发技术的地位，但由于国内开源的相对封闭，造成了新手学习PHP的各种障碍。例如，国内缺少针对入门读者的系统教材，大部分读者只能借助PHP手册学习PHP。本书力争打破这种局面，帮助入门读者系统掌握PHP技术。

本书系统整理和分析了PHP技术，合理划分篇章结构，帮助读者建立完善的PHP技术体系。同时，针对PHP技术新的应用趋势，着重讲解了新应用所使用的技术。考虑到新手入门的特点，本书有针对性地结合了大量示例，帮助读者尽可能好地掌握每项技术。同时为了方便读者可以高效而直观地掌握PHP技术，本书提供了全程多媒体教学视频，以辅助读者学习本书所讲解的内容。

本书有何特色

1．配多媒体教学视频

本书提供配套多媒体教学视频辅助教学。视频涵盖本书各个知识点，从而帮助读者高效、直观地掌握各项技术。

2．门槛低，容易入门

PHP之所以非常热门的原因之一就是非常容易入门。因此，不要求读者有太多基础，只要跟着内容的讲解一步步走下去，就可以很容易地做出自己的页面。

3．内容全面、系统

本书详细介绍了PHP开发所需要的知识，包括语法、函数、数组、对象等，还特别介绍了网站模版和常用模块，通过学习这些技术，读者就可以轻松开发PHP页面。

4．讲解由浅入深，循序渐进

本书的编排采用循序渐进的方式，内容梯度从易到难，讲解由浅入深，适合各个层次的读者阅读，并均有所获。

5．写作细致，处处为读者着想

本书内容编排、概念表述、语法讲解、示例讲解、源代码注释等都很细致，作者讲解时不厌其烦，细致入微，将问题讲解得很清楚，扫清了读者的学习障碍。

6．贯穿大量的开发实例和技巧

本书在讲解知识点时贯穿了大量短小精悍的典型实例，并给出了大量的开发技巧，力求让读者获得真正实用的知识。

7．提供教学 PPT，方便老师教学

本书适合大中专院校和职业学校作为职业技能的教学用书，所以专门制作了教学 PPT，以方便各院校的老师教学时使用。

本书内容安排

第 1 篇　PHP 语言篇（第 1～7 章）

本篇主要内容包括：PHP 概述、PHP 数据类型与运算符、语言结构、函数、数组、面向对象编程以及错误处理。本篇主要是让读者了解 PHP 的应用领域、运行环境及 PHP 的基础语法知识。该部分是 PHP 的基础，只有完全掌握了本篇的知识，才可以在后面的篇幅中游刃有余。

第 2 篇　PHP 技术篇（第 8～12 章）

本篇主要内容包括：字符串处理函数、文件系统操作、图像处理、数据库管理系统以及 Cookie 和 Session。本篇是 PHP 的进阶部分，主要介绍了 PHP 中相关处理的技术。学习本章后读者应该有能力使用 PHP 实现一些简单的功能。

第 3 篇　PHP 实战篇（第 13、14 章）

本篇主要内容包括：网站模版和常用模块。本篇介绍的是 PHP 的实际应用，读者学习完本章的知识后，可以搭建起一个功能完整的论坛程序和个人网站。

本书光盘内容

- ❑ 本书配套多媒体教学视频；
- ❑ 本书实例涉及的源代码。

本书读者对象

- ❑ Web 服务器端开发入门人员；
- ❑ 网页维护人员；
- ❑ 网站建设和开发人员；
- ❑ 网站制作爱好者；
- ❑ 网站制作培训机构人员；

- ❑ 大中专院校的学生。

本书阅读建议

- ❑ 建议没有基础的读者，从前到后顺次阅读，尽量不要跳跃。
- ❑ 书中的实例和示例建议读者都要亲自上机动手实践，学习效果更好。
- ❑ 课后习题都动手做一做，以检查自己对本章内容的掌握程度，如果不能顺利完成，建议回过头来重新学习一下本章内容。
- ❑ 学习每章内容时，建议读者先仔细阅读书中的讲解，然后再结合本章教学视频，学习效果更佳。
- ❑ 部分内容讲解涉及 Web 核心组件或者系统组件，建议读者在操作之前，备份相应文件，避免对系统造成不必要的影响。

本书作者

本书由张少卓主笔编写。其他参与编写的人员有毕梦飞、蔡成立、陈涛、陈晓莉、陈燕、崔栋栋、冯国良、高岱明、黄成、黄会、纪奎秀、江莹、靳华、李凌、李胜君、李雅娟、刘大林、刘惠萍、刘水珍、马月桂、闵智和、秦兰、汪文君、文龙、陈冠军、张昆。

阅读本书的过程中，若有任何疑问，可以发邮件到 book@wanjuanchina.net 或 bookservice2008@163.com，或者到 www.wanjuanchina.net 的图书论坛上留言，以获得帮助。

编著者

目　　录

第 1 篇　PHP 开发基础

第 2 篇 PHP 开发进阶

第 3 篇　PHP 开发实战

第 1 篇　PHP 开发基础

第 1 章　PHP 概述

PHP 是一门编程语言，它像我们熟知的 C 语言、C++语言一样，可以用来编写出应用程序。同时，PHP 也是一种现在非常热门的动态网站开发技术。PHP 语言是开源的，由于这个特点，使 PHP 成为了当下主流的网站开发技术，使用 PHP 开发的网站遍及各个行业领域。在具体开始学习 PHP 之前，本章首先讲解动态网站开发技术，便于读者对 PHP 有个初步的了解，以及为之后的学习做好规划。

1.1　动态网站技术

我们日常看到的网站都是大同小异的，以专业的角度看，网站大致可以分为两个大的种类——静态网站和动态网站。现在的网站大多是动态网站，下面主要来讲解一下动态网站的构成和所应用的技术。

1.1.1　什么是动态网站

动态网站并不是指具有动画功能的网站，一般是指与网页与服务器有数据交互的网站，最明显的特点就是一般是实时的。就像我们看一些新闻网站的时候，新闻内容是随时间改变而变化的。动态网站除了要设计我们所看到的网页外，还要通过一些技术来使网站具有更多自动的和高级的功能。PHP 语言就常用来做这些事情。

接下来我们来了解一下动态网站的运行机制，整个过程如图 1.1 所示。首先，用户在

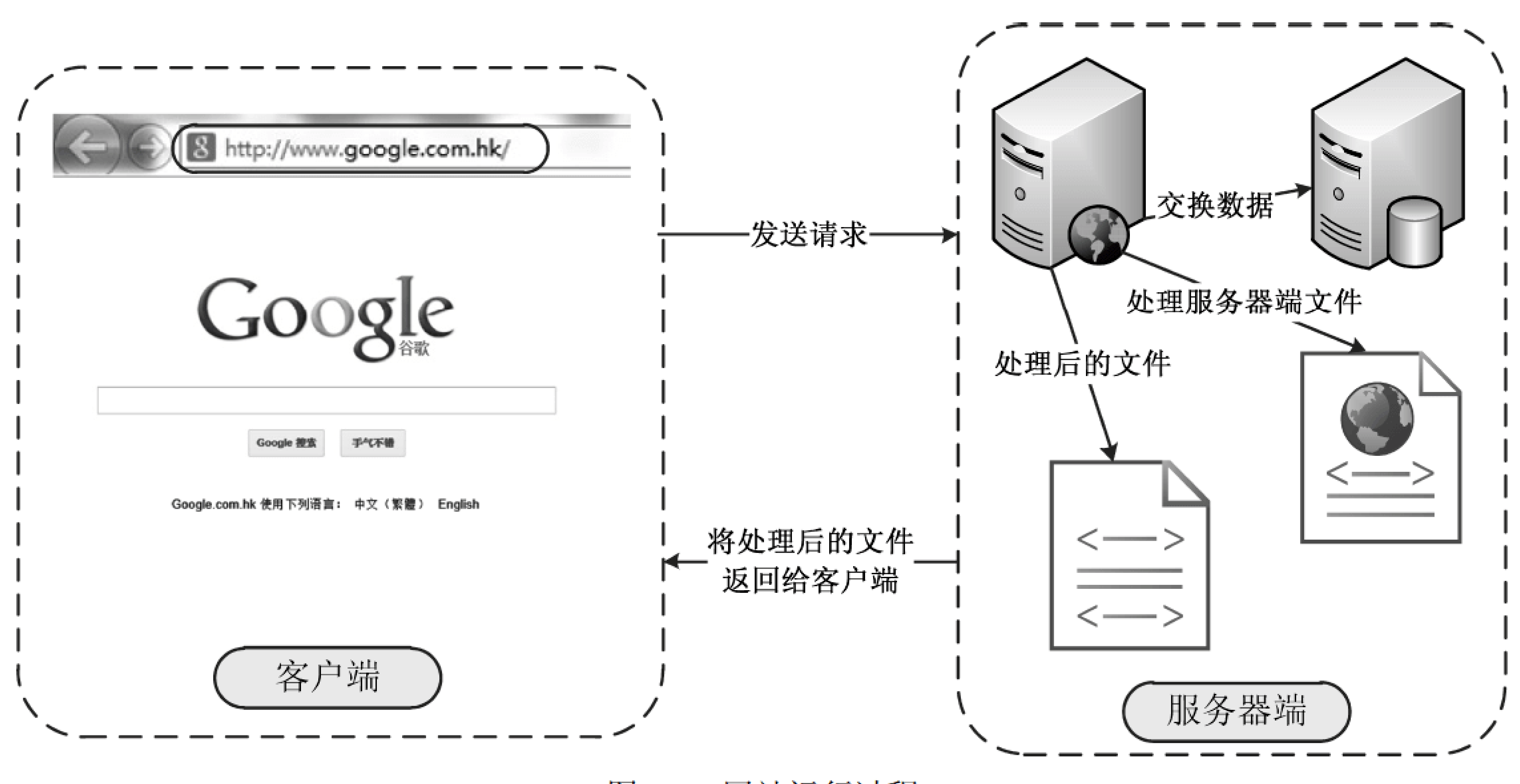


图 1.1　网站运行过程

客户端浏览器地址栏中输入网址并提交；浏览器获取到网址后会查找该网址对应的服务器；服务器接受请求后，找到服务器中对应的网页文件，进行处理（通常是与数据库服务器交互数据）；然后，服务器将处理的结果返回到客户端，最后客户端通过浏览器将网页内容显示给用户。

其中，客户端可以是用户使用的计算机，也可以是移动终端，可以说只要有一个可以联网的浏览器。服务器是管理网站资源并为用户提供服务的计算机软件，也可以称为是后台。当接收到客户端发送过来请求时，服务器会找到客户端请求的文件用相应技术进行处理，然后将处理后的网页代码返回给客户端显示。

1.1.2　前台技术

在 1.1.1 节中我们已经清楚地了解了一个网站的运行过程。在客户端浏览器所呈现出来的内容我们称之为网页的前台。网页的前台可以起到宣传、广而告之的作用，通常会做得比较华丽或者新颖。前台技术一般由 HTML、CSS 和 JavaScript 这 3 大技术构成。下面我们就来简单了解一下这些技术。

1. HTML

HTML 的英文全称是 HyperText Markup Language，中文含义即“超文本标记语言”。它是一种简单、通用的标记语言。很多网页都是通过 HTML 格式来返回的，我们可以通过查看一个网页的属性来查看这个网页类型，如图 1.2 所示。



图 1.2　网页属性

虽然我们可以看到类型是 HTML 文档，但它通常不是只由 HTML 技术编写的。简单地说，HTML 只用来控制网页文本的显示，其他的例如布局和一些效果是由别的技术来完成的。

这里我们拿非常简单的一段代码来展示一下 HTML 技术。如图 1.3 所示，其中第 4 行代码控制网页标题文字，第 8 行、10 行代码用于控制网页主体中的文字。由于本书的侧重不在 HTML 技术，因此我只做简单讲解。如果读者想学习更多的 HTML 可以参考相关书籍。

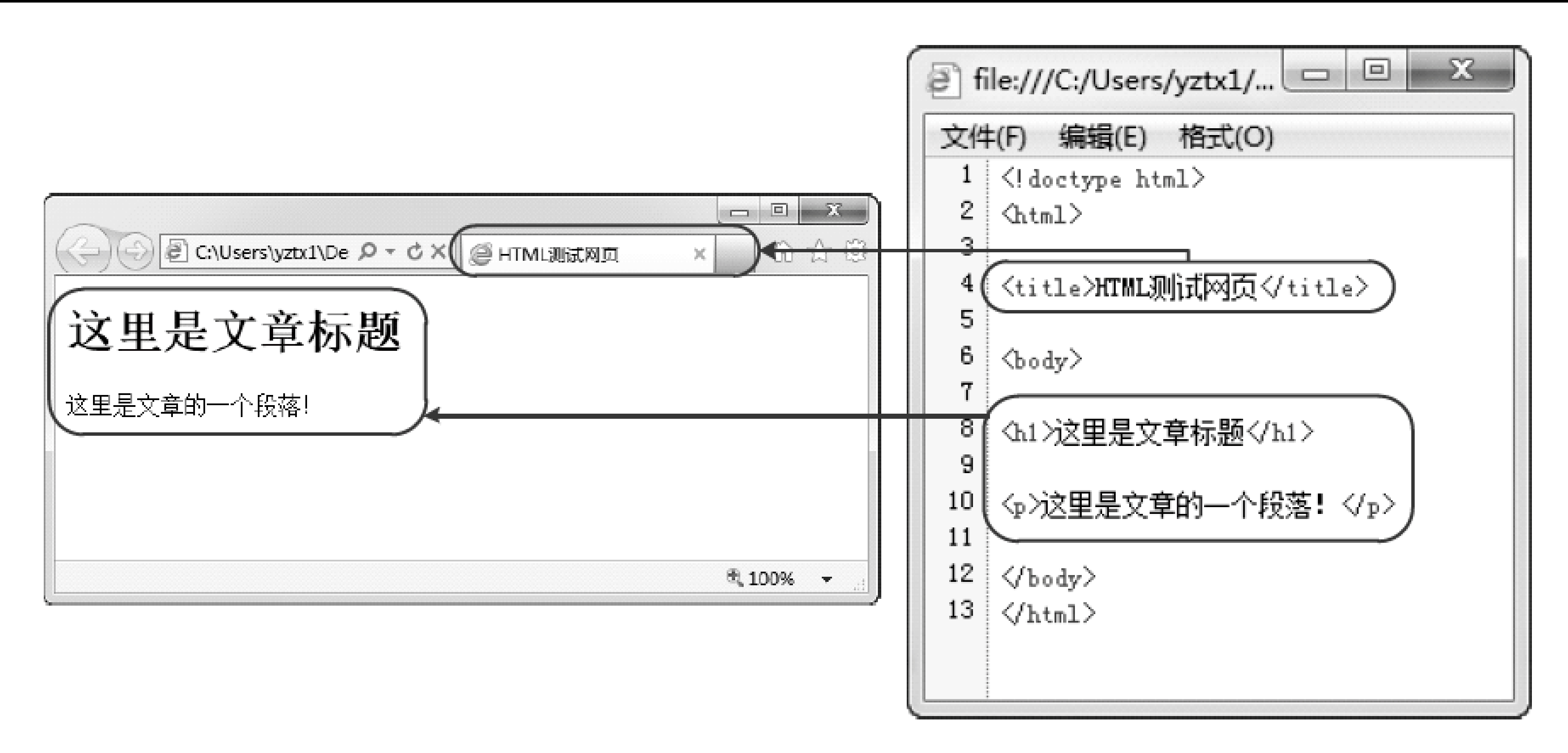


图 1.3　HTML 技术

2. CSS 技术

CSS 的英文全称是 Cascading Style Sheets，中文含义即“层叠样式表”。它一般用来在网页中控制和修饰一些 HTML 元素的显示方式。这里我们仍以一个简单的示例来做讲解。如图 1.4 所示，图中框中的代码即是 CSS 代码，而浏览器中显示不同元素的背景颜色就是这段 CSS 代码修饰的结果。

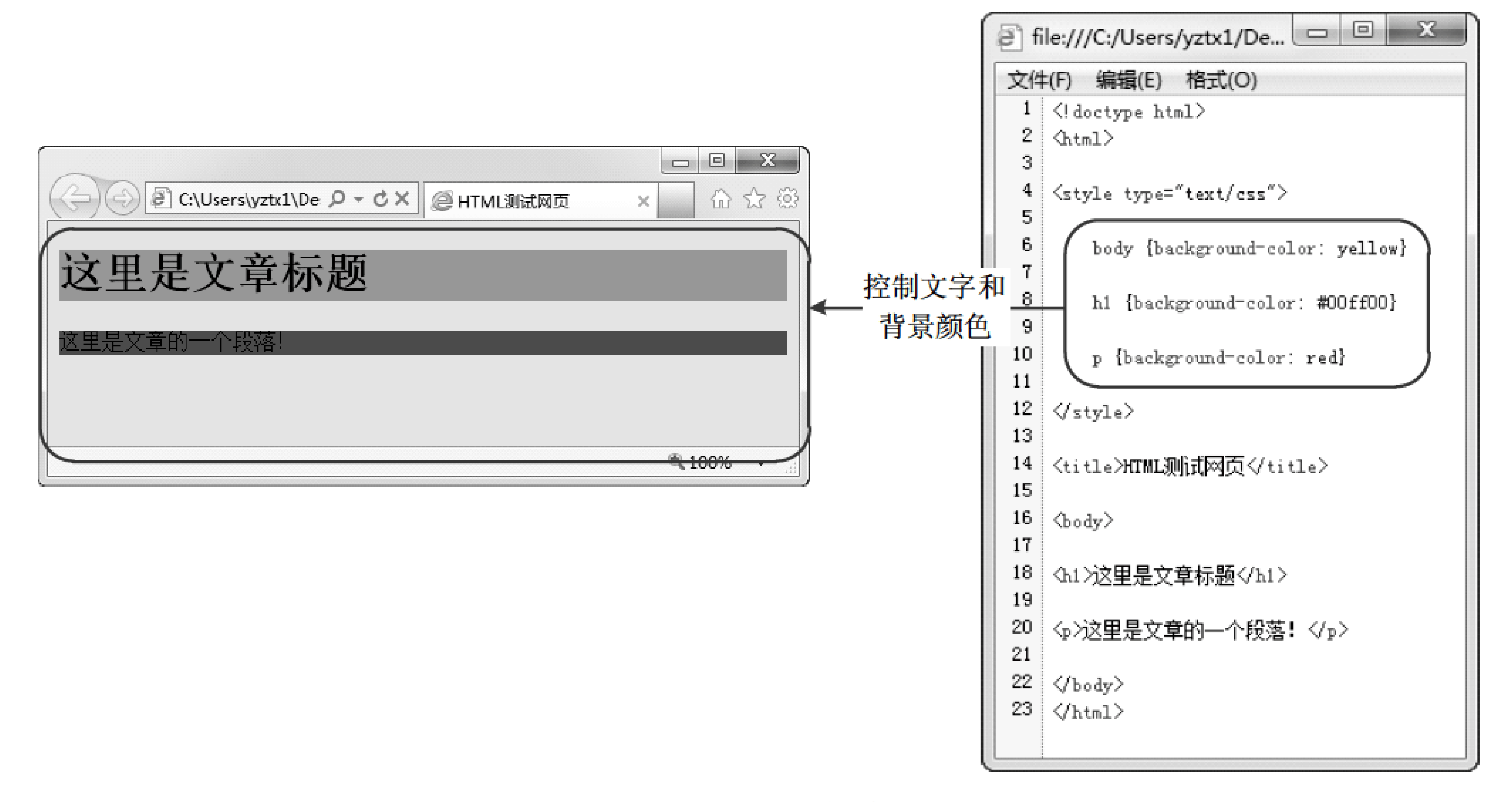


图 1.4　CSS 技术

同样地，如果读者想要学习更多的 CSS 知识，可以参考相应的书籍，这里就不再做详细讲解。

3. JavaScript 技术

JavaScript 是一种脚本语言，在网页技术中被设计用来向 HTML 页面添加动态交互的效果。这里我们以一个简单的显示当前时间的 JavaScript 代码来做讲解，如图 1.5 所示。

在图 1.5 中就利用 JavaScript 代码来显示当期的时间，然后由浏览器解析显示在页面中。在书本上我们不会体验到它的特点。在实际中，当我们刷新浏览器的时候，时间会随时更新，而 JavaScript 代码是不需要做任何改变的。



图 1.5　JavaScript 技术

1.1.3　后台技术

网站的后台，就是服务器端处理客户端发送过来的请求。现在常用的网站后台技术有 ASP、ASP.NET、JSP、PHP 及与数据库的结合等。接下来我们主要了解一下 PHP 和数据库。

1. PHP

PHP 是 PHP：Hypertext Preprocessor 的缩写，中文含义即“PHP 超文本预处理语言”。PHP 作为一种嵌入 HTML 的脚本语言，运行于服务器端。我们仍然使用 PHP 来实现显示当前时间的功能，代码及运行结果如图 1.6 所示。我们可以看到使用 PHP 也可以实现类似的效果。

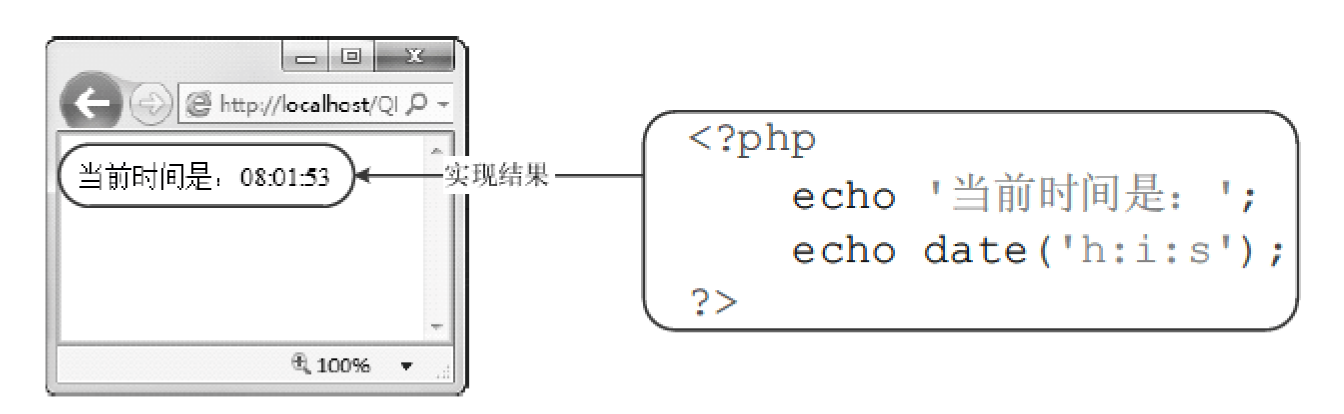


图 1.6　PHP 实现显示当前时间

当然，真正的从代码到显示我们期待的结果是远没有这么简单的，其中的过程我们将在学习过程中逐步了解和掌握。

2. 数据库

数据库可以理解为网站存放数据的一个仓库，可以说一个网站脱离了数据库是不能存活的。现在比较流行的数据库有 MySQL、SQL Server 和 Oracle 数据库等。数据库里存放着网站的很多数据。网站可以通过编程语言对数据库进行数据的增加、删除和修改等操作。我们将要学习的数据库部分就是使用 PHP 来操作的。

1.2　构建 PHP 环境

一段代码要运行起来，就必须要有一个供这段代码运行的一个环境。例如我们前面了解的 HTML 代码的运行环境就可以是浏览器。而 PHP 要完整地运行起来就没有那么简单

了。因此，本节我们要讲解的就是为我们将要学习的 PHP 搭建起一个它的“生存环境”。

1.2.1　PHP 开发环境

构成一个 PHP 开发环境一般情况下主要由服务器程序、编程语言核心和数据库程序 3 部分组成。三者互相配合，完成整个动态网站功能的流程。服务器程序一般选用 Apache 服务器，数据库选用 MySQL，编程语言当然是 PHP。

Apache 是一个 Web 服务器，PHP 是一种 HTML 内嵌式的编程语言，MySQL 是一个开源的数据库管理系统。它们的官方网址及最新版本如下：

- Apache 的官方网站是 http://www.apache.org/，目前最新版本为 Apache 2.4.3。
- PHP 的官方网站是 http://www.php.net/downloads.php，目前最新的版本为 PHP 5.4.8。
- MySQL 的官方网站是 http://www.mysql.com/downloads/，最新版本为 MySQL 5.5。

对于初学者来说，分别下载和配置这些工具是一件比较费力的事情，很多对 PHP 感兴趣的人都因为运行环境的配置而放弃。这里读者无须担心，对于学习来说，配置不是非常重要的事，因此我们将采用集成开发环境来搭建 PHP 运行环境。

1.2.2　XAMPP 集成开发环境

现在常用的集成开发环境有很多，我们采用的是 XAMPP 集成开发环境。XAMPP 是由 Apache Friends 开发的一个将 Apache 服务器与 phpMyAdmin、PHP 及 MySQL 等软件集合在一起的安装包。它不仅可以在 Windows 平台下安装，而且可以在 UNIX 平台及类 UNIX 平台下安装。下面我们就从零开始搭建一个完整的 PHP 运行环境。

1. 下载 XAMPP

下载最新版本的 XAMPP，可打开其官方网站地址 http://www.apachefriends.org/zh_cn/xampp.html，下载适用于 Windows 的 XAMPP，操作如图 1.7 所示。

该官方网站提供 3 种格式的安装包：第一种是二进制的安装包，第二种是 ZIP 格式的压缩包，第三种是 7-ZIP 格式的自解压包。读者可自行选择下载哪一种格式。我们这里使用的是二进制的安装包。

图 1.7　XAMPP 下载

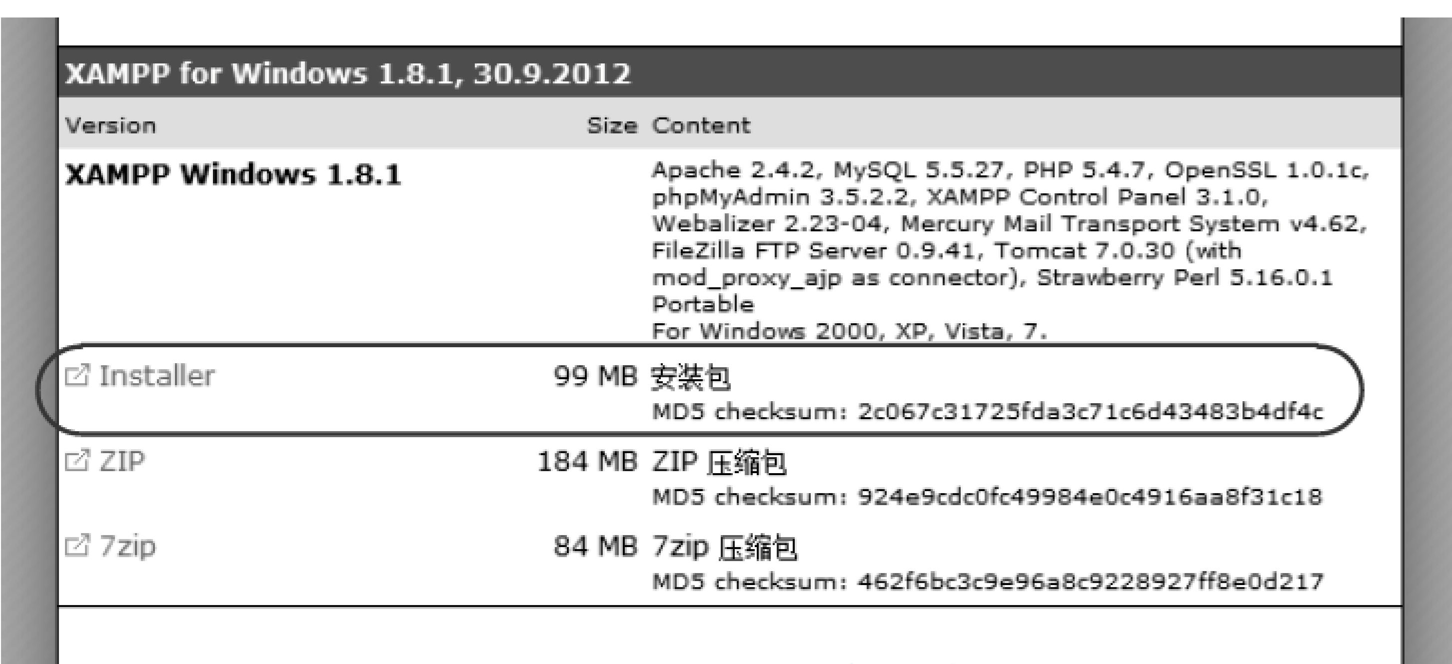
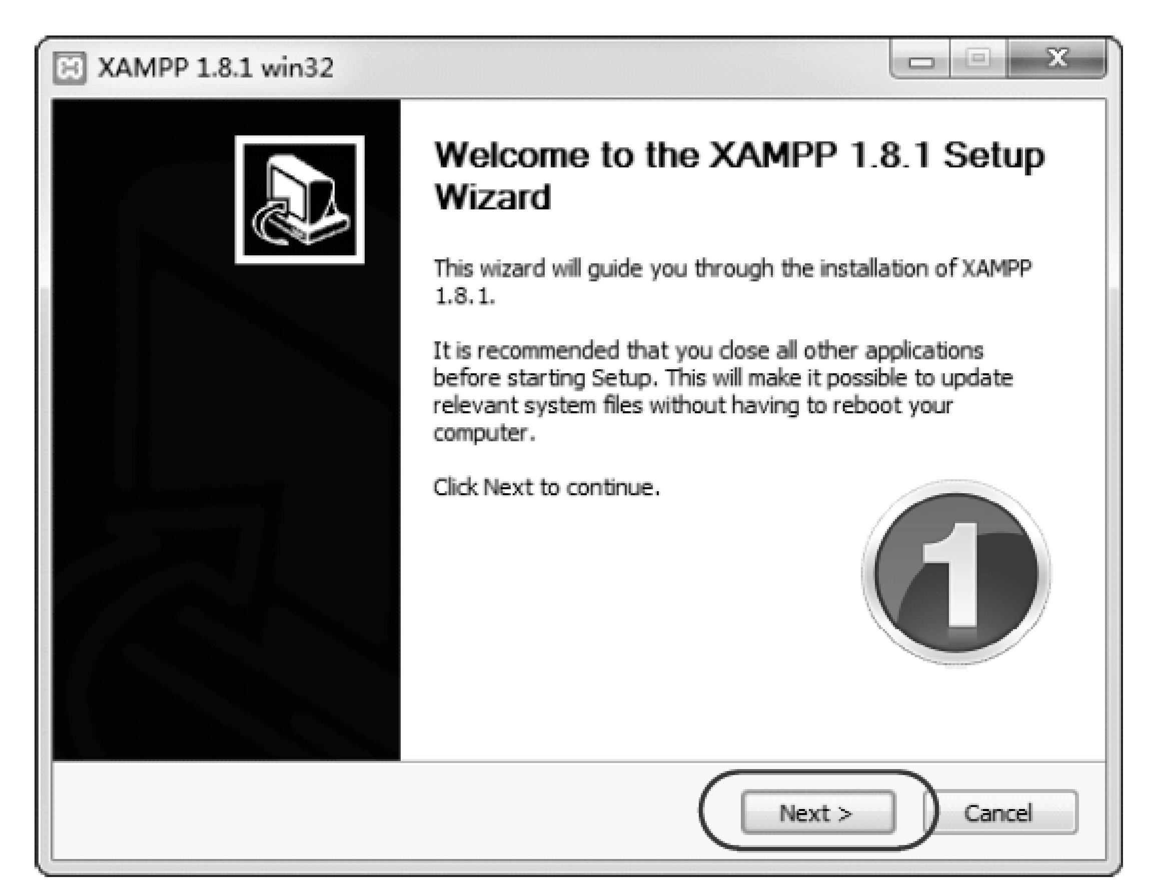
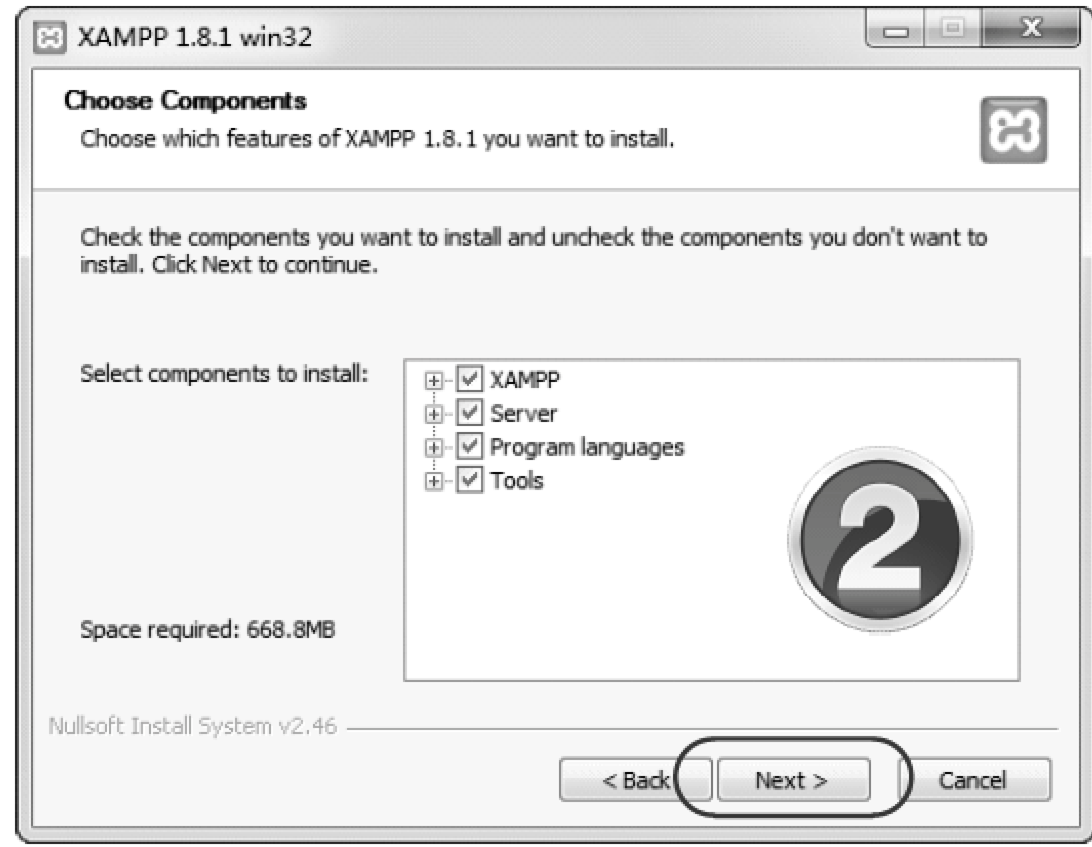


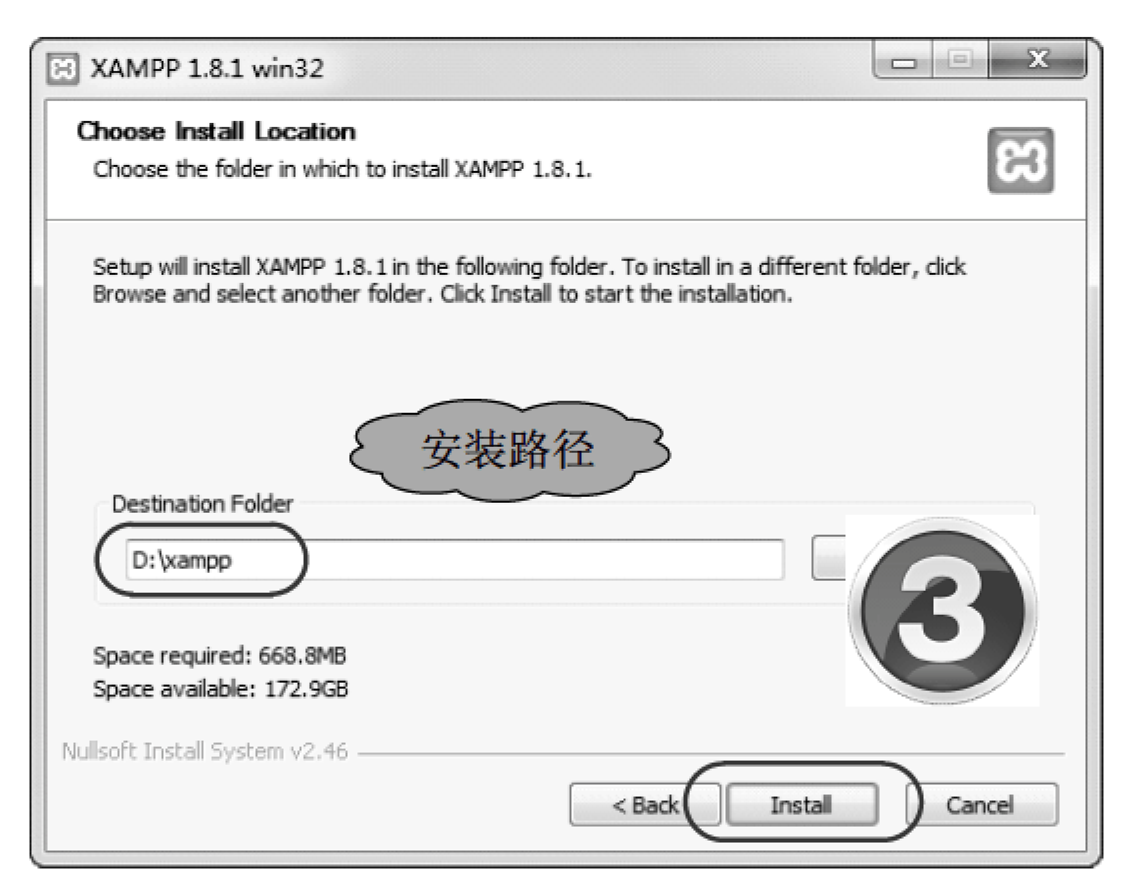
图 1.7　XAMPP 下载（续）

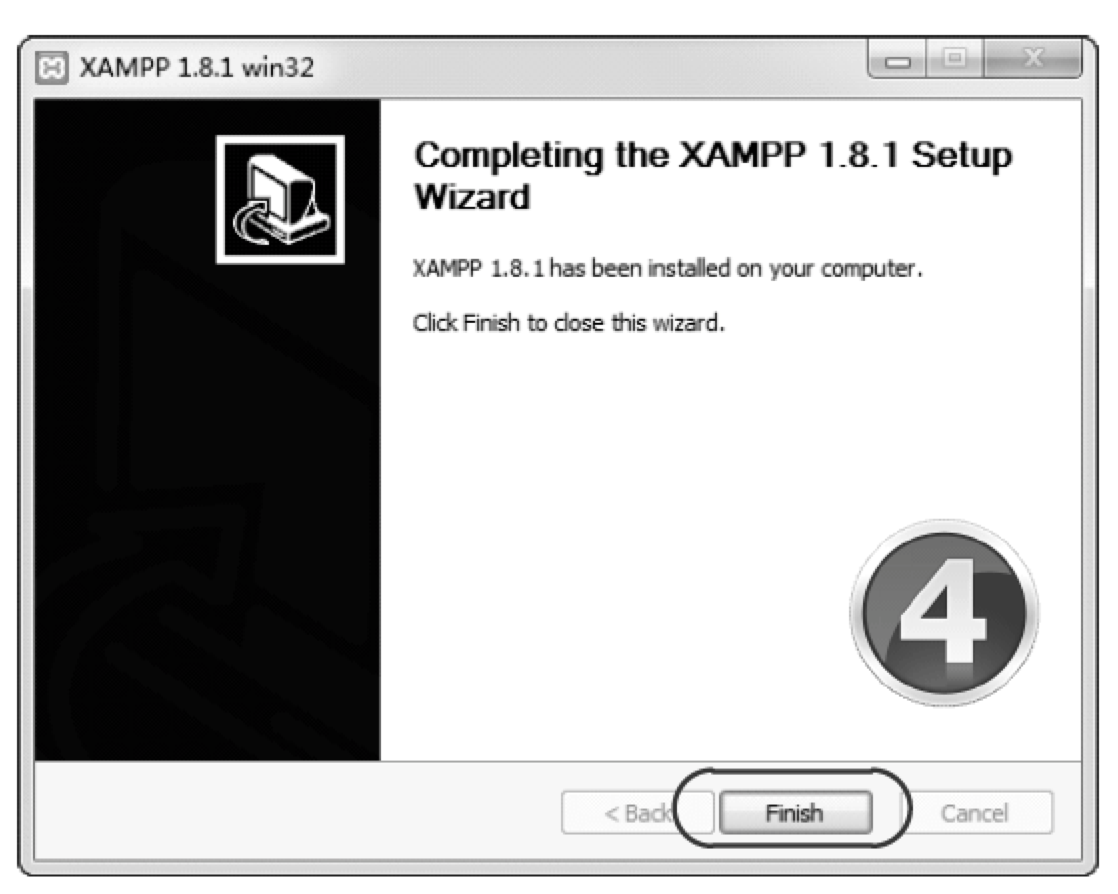
2. 安装 XAMPP

将下载的文件直接双击打开，安装程序就会一步步指导我们安装，详细安装过程如图 1.8 所示。









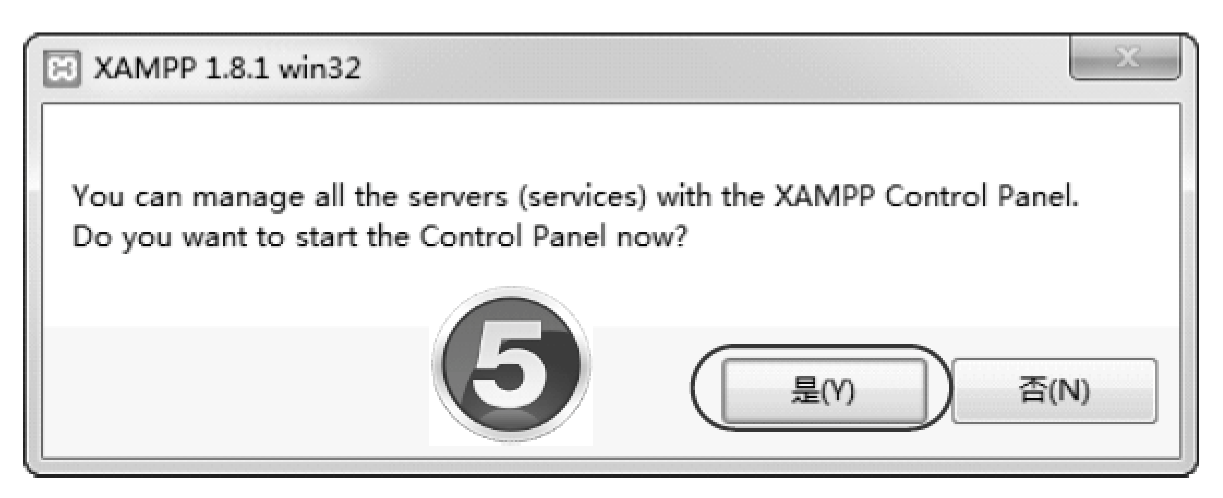


图 1.8　XAMPP 详细安装过程

3. 运行运行 XAMPP

在图 1.8 中所示的步骤进行到第 4 步时安装已经完成了，第 5 步就是询问我们是否现在启动 XAMPP 的控制面板。我们单击“是”以启动 XAMPP 的控制面板，如图 1.9 所示。

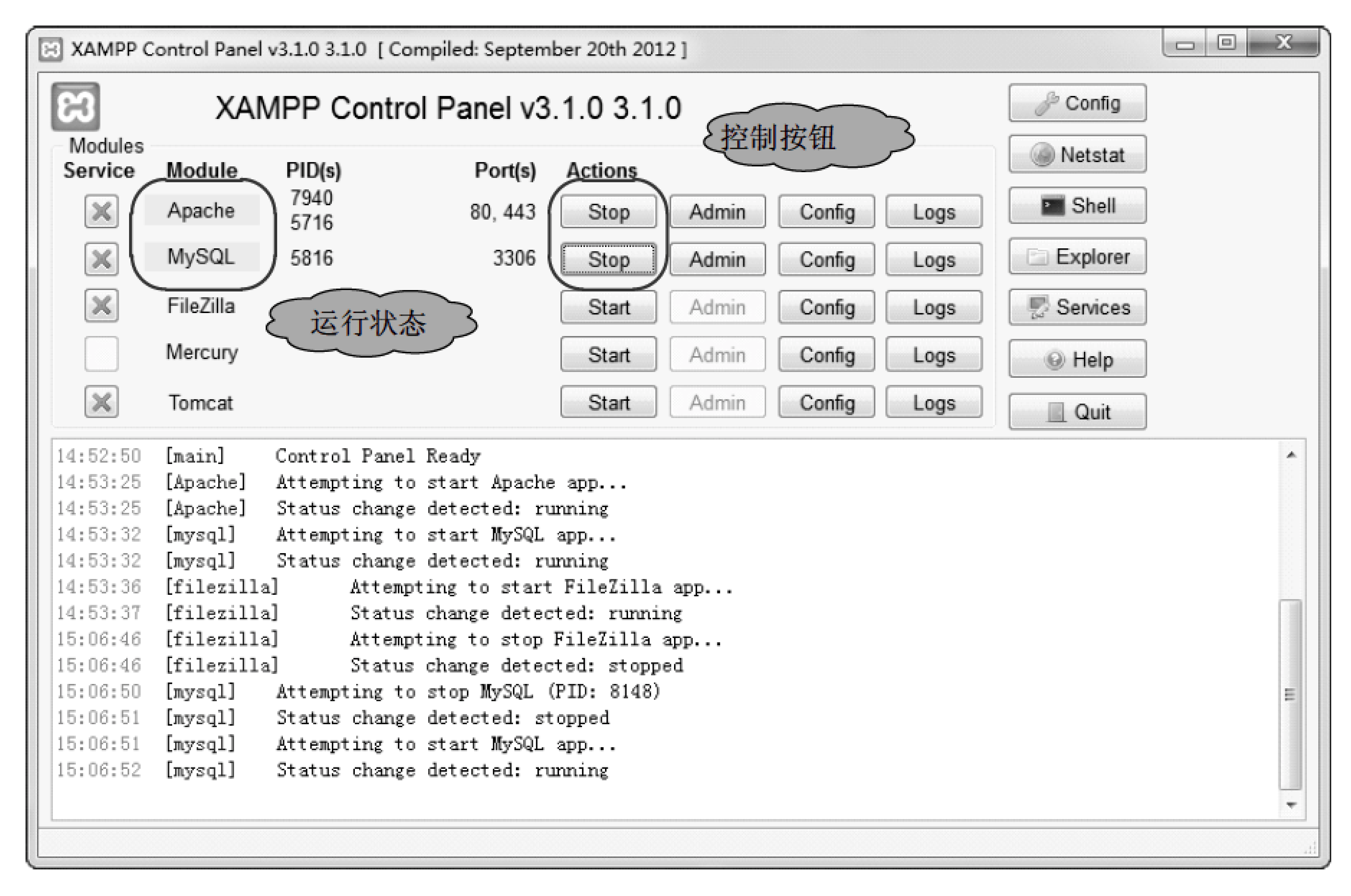


图 1.9　Apache 与 MySQL 的启用状态

可以在 XAMPP 控制面板中使用控制按钮来控制每个模块。对应模块背景变为青色则表示该模块已经成功运行了。

可以通过在浏览器地址栏中输入 http://localhost/并提交来验证，出现如图 1.10 所示的页面即表示 XAMPP 安装成功。



图 1.10　验证 XAMPP 安装

对于在其他平台下 XAMPP 集成环境的搭建我们这里就不做讲解，其安装方法可在 XAMPP 官方网址查看。

1.3　第一个程序 Hello World

前面的小节中我们已经安装好了 PHP 的运行环境，并且通过一个网做了验证，但那总归是不是自己写的。本节中，我们就来做世界上每个新诞生的语言都要做的事情——输出“Hello,World!”。

1.3.1　代码编写工具的选择

PHP 编写可以使用非常简单的工具，例如记事本、Notepad2、EDITPLUS、NOTEPAD++等简单轻巧的工具。也可以使用 Zend Studio 这样功能强大的集成开发环境。作为初学者来说，在没有辅助功能的情况下编写代码是最能锻炼能力的。因此我们提倡读者使用一些轻量级的文本编辑器，来作为自己学习 PHP 的工具。下面我们简单介绍一些轻量级的编辑工具。

1. 系统自带记事本程序（Notepad）

系统自带的记事本程序是我们最熟悉的文本编辑工具了，界面如图 1.11 所示。

kuang.html - 记事本

文件(F)　编辑(E)　格式(O)　查看(V)　帮助(H)

```
<!DOCTYPE>  <html>  <head>  <meta charset="utf-8">  <title></title>
<script type="text/javascript">       var context;        var desPos=
{x:50,y:50};          var startMovingPos = {x:50,y:50};        var rectWidth =
100,rectHeight = 100;        var canvas;        var timer = 0;//计时器ID
    function MouseDown(e){                 e = e || window.event;        if
(1 == e.which){//left button                     clearInterval(timer);
      desPos.x = e.clientX - canvas.scrollLeft;              desPos.y =
e.clientY - canvas.scrollTop;                  timer = setInterval(draw,0.5);
              }               }             function DrawRect(){            canvas =
document.getElementById("view");               context = canvas.getContext
("2d");              context.fillStyle = "rgb(250,123,45)";
context.fillRect(desPos.x,desPos.y,rectWidth,rectHeight);            }
function draw(){               context.clearRect
(0,0,canvas.clientWidth,canvas.clientHeight);              var delX = desPos.x
- startMovingPos.x;           var delY = desPos.y - startMovingPos.y;
   startMovingPos.x += delX/10;             startMovingPos.y += delY/10;
      context.fillRect
(startMovingPos.x,startMovingPos.y,rectWidth,rectHeight);         }
</script>  </head>  <body onload="DrawRect()">  <canvas id="view"
width="640" height="560" style="background: silver;"
onmousedown="MouseDown()"></canvas>  </body>  </html>
```

图 1.11　记事本编辑器

记事本的特点是轻巧、使用方便、通常系统自带，无须自己安装。

2. Notepad2 文本编辑工具

Notepad2 是一个外观类似系统记事本的文本编辑工具，但比记事本有了很大的增强，它的界面如图 1.12 所示。

Notepad2 相对记事本来说就比较高级了，最明显的特点是具有了显示行号、内建各种程序语法的高亮度显示、改变背景颜色、支持 Unicode 与 UTF-8 的功能。

3. Notepad++代码编辑器

Notepad++是一款 Windows 环境下免费开源的代码编辑器，它的界面如图 1.13 所示。Notepad++不仅有 Notepad2 的语法高亮度显示，并且加入了语法折叠功能，而且支持

宏及扩充基本功能的外挂模块，这就使得 Notepad++可以扩展为非常强大的编辑器。

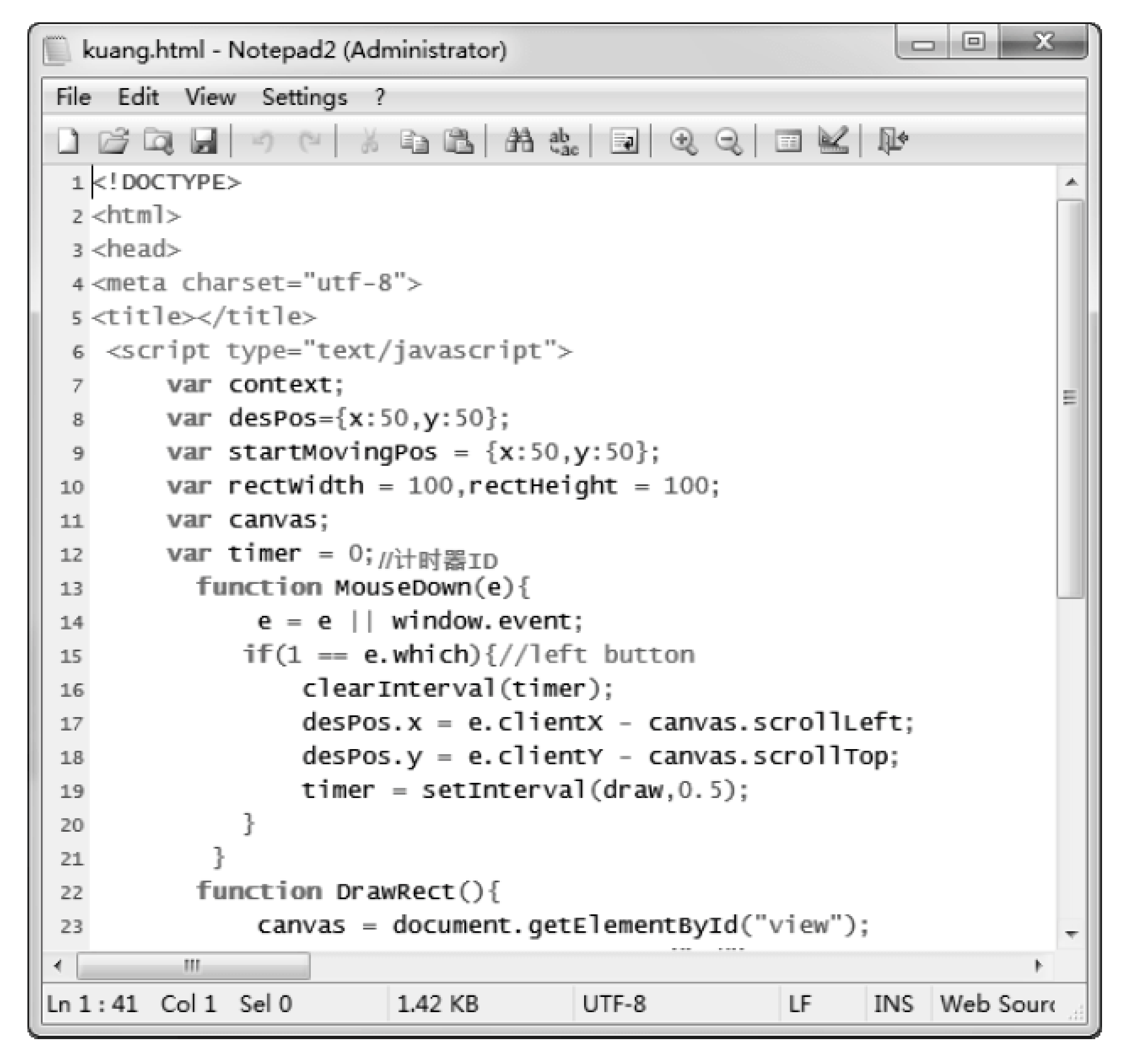


图 1.12　Notepad2 界面

```
<!DOCTYPE>
<html>
<head>
<meta charset="utf-8">
<title></title>
 <script type="text/javascript">
     var context;
     var desPos={x:50,y:50};
     var startMovingPos = {x:50,y:50};
     var rectWidth = 100,rectHeight = 100;
     var canvas;
     var timer = 0;//计时器ID
       function MouseDown(e){
           e = e || window.event;
          if(1 == e.which){//left button
              clearInterval(timer);
              desPos.x = e.clientX - canvas.scrollLeft;
              desPos.y = e.clientY - canvas.scrollTop;
              timer = setInterval(draw,0.5);
          }
       }
       function DrawRect(){
```

图 1.13　Notepad++界面

4. VIM 编辑器

VIM 是从 vi 发展出来的一个文本编辑器。代码补完、编译及错误跳转等方便编程的功能特别丰富，在程序员中被广泛使用，和 Emacs 并列成为类 Unix 系统用户最喜欢的编辑器，它的界面如图 1.14 所示。

以上就是我们为读者推荐的一些免费而且小巧的可以用来编写 PHP 代码的工具。读者可以根据自己的喜好选择一款来使用。前面 3 款相对来说使用没有特别的不适应之处。第 4 款 VIM 编译器功能非常强大，而且熟练后工作效率会很高但是要上手则需要一定的时间。因此，读者要充分了解这个编辑器后再选择。本书中采用的是 Notepad++编辑器，但是读者可以完全放心地选择自己喜欢的编辑器，在学习中它们之间不会有丝毫的差异。

```
kuang.html (~\Desktop) - GVIM
文件(F) 编辑(E) 工具(T) 语法(S) 缓冲区(B) 窗口(W) 帮助(H)
<!DOCTYPE>
<html>
<head>
<meta charset="utf-8">
<title></title>
 <script type="text/javascript">
     var context;
     var desPos={x:50,y:50};
     var startMovingPos = {x:50,y:50};
     var rectWidth = 100,rectHeight = 100;
     var canvas;
     var timer = 0;//璁℃椂鍣↖D
       function MouseDown(e){
           e = e || window.event;
          if(1 == e.which){//left button
              clearInterval(timer);
              desPos.x = e.clientX - canvas.scrollLeft;
              desPos.y = e.clientY - canvas.scrollTop;
              timer = setInterval(draw,0.5);
          }
        }
       function DrawRect(){
           canvas = document.getElementById("view");
           context = canvas.getContext("2d");
"~\Desktop\kuang.html" [noeol][unix] 41L, 1459C          1,1          顶端
```

图 1.14　VIM 编辑器界面

1.3.2　编写第一个程序 Hello world!

在选择好一个编辑器后，我们就来写出我们的第一个 PHP 程序。

1. 进入 Apache 服务器主目录

首先读者要做的是找到 Apache 服务器的主目录。它是 XAMMP 安装路径下的 htdocs 文件夹，安装路径是在 XMMAP 安装中第 4 步设置的。例如安装路径是 D:\xammp 那么服务器主目录就是 D:\xammp\htdocs，如图 1.15 所示。



图 1.15　Apache 服务器主目录

2. 创建 PHP 文件

在进入到 Apache 主目录后，就在该文件夹下创建一个文本文档并将其后缀改为“.php”，如图 1.16 所示。

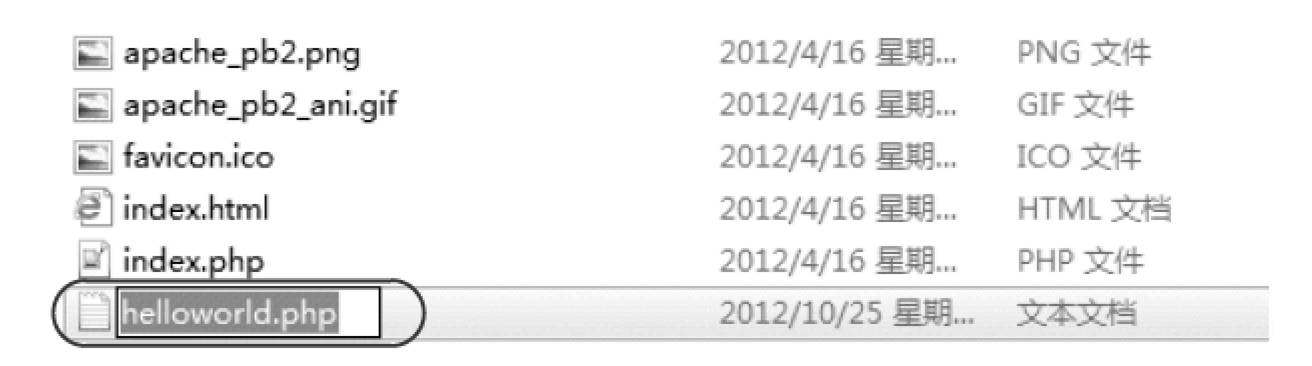


图 1.16　创建 PHP 文件

注意：注意有的系统是隐藏后缀名的，那样会导致创建的文件不是 PHP 文件。解决办法就是改变一项设置。操作流程如下：

（1）单击开始按钮，选择控制面板。

（2）打开文件夹选项，选择“查看”选项卡。

（3）将“隐藏已知文件类型的扩展名”前面的复选框取消，单击“确定”按钮保存设置。

设置后显示如图 1.17 所示。

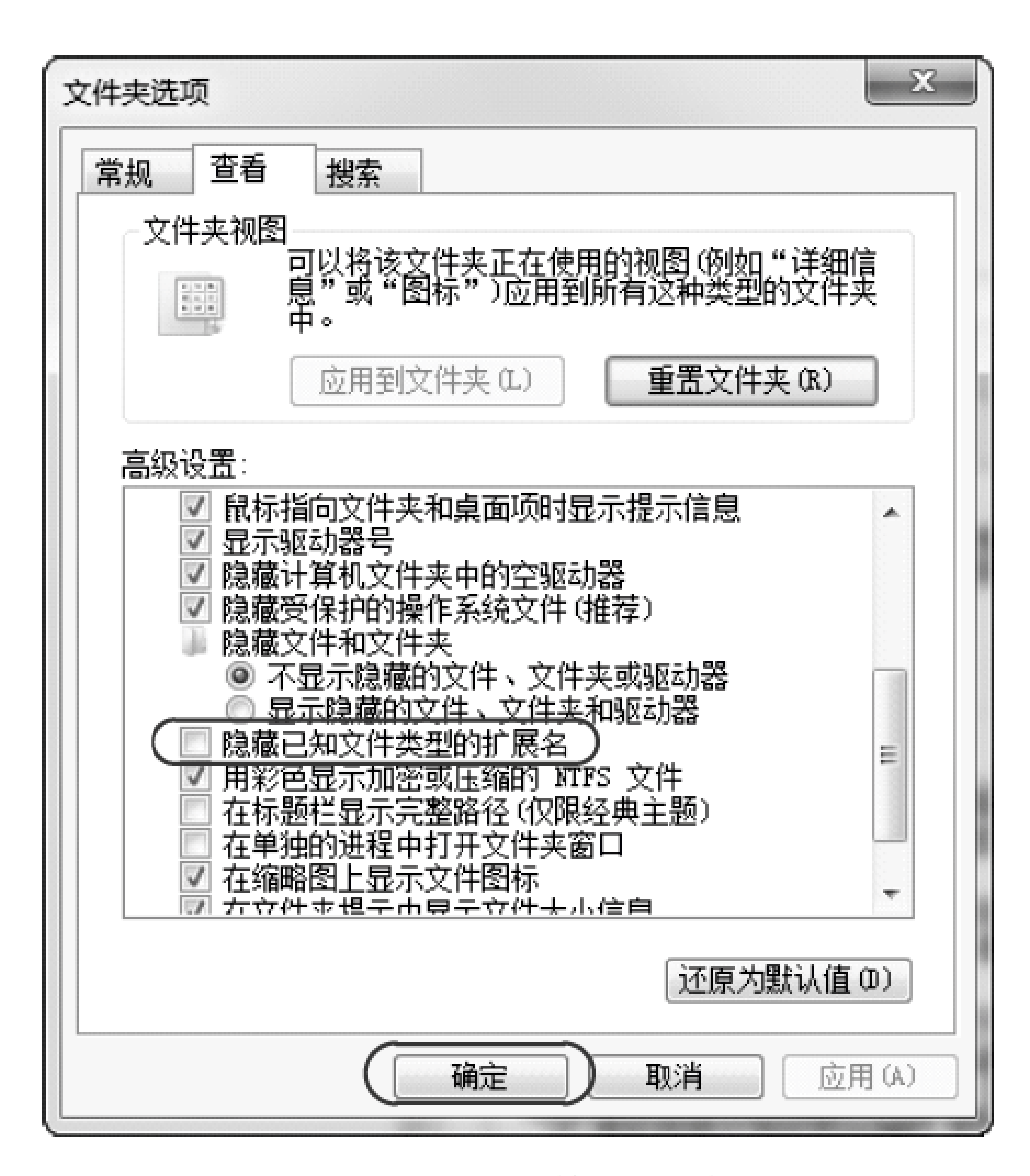


图 1.17　设置文件扩展名显示

3. 关联打开程序

在创建好 PHP 文件后，该文件默认没有关联打开该文件的程序。我们需要将 PHP 文件关联打开程序设置为我们喜欢的工具，设计步骤如图 1.18 所示。

4. 编写代码

在关联好打开程序之后，我们就可以双击该文件来打开，然后在文件中写入如下代码。

```
<?php
```

```
    echo "Hello World! ";
?>
```

图 1.18　关联默认程序步骤

写入完成后，保存该文件（快捷键为 Ctrl+S），如图 1.19 所示。



图 1.19　hello world 代码

这段代码会将“Hello World!”字符输出至浏览器。

5. 运行代码

因为我们的代码文件已经放在了 Apache 服务的主目录中，服务器已经可以正确找到该文件。因此可以直接在浏览器地址栏中输入 http://localhost/helloworld.php 并提交，我们将会看到如图 1.20 所示的执行结果。

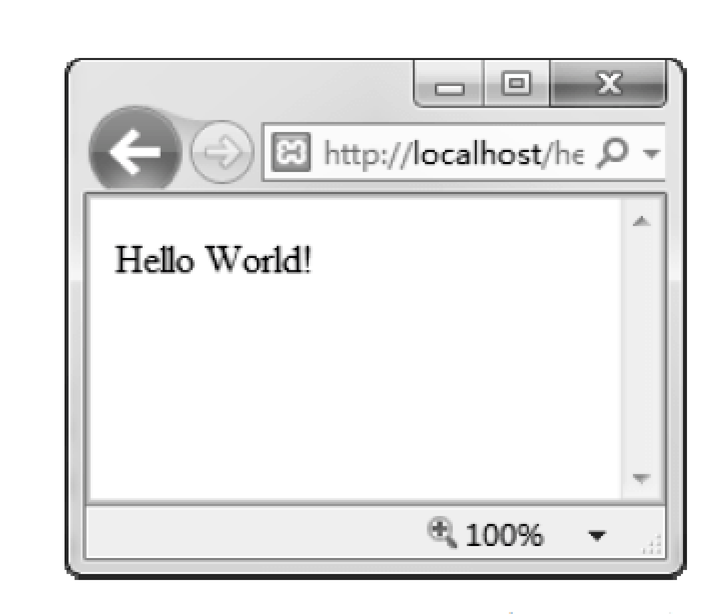


图 1.20　helloworld 程序运行结果

程序成功执行后，也就预示着我们已经简单体验了一个完整的 PHP 代码运行流程。这里有了一个很好的开始，在接下来的章节中，我们将一步步来了解、认识和熟练使用 PHP。

1.4　小　　结

本章主要讲解了网站的一些常用的技术，然后我们选择了 PHP 来作为我们学习的技术。之后通过安装 XAMMP 集成开发环境，在满足学习使用要求的同时避免了复杂的环境配置打击读者学习的信心。接着讲解了一个简单的 PHP 程序从创建到成功运行的每个细节，努力做到让读者没有一处不懂，对后面的学习充满信心。

1.5　本 章 习 题

1. 完全安装运行一次 XAMPP 集成环境。
2. 找到自己安装的 XAMPP 环境的根目录。
3. 安装一个自己喜欢的编辑器。
4. 将“PHP,I'm coming!”显示在自己的浏览器中。

第 2 章　PHP 数据类型与运算符

学习任何一门语言都是要从基础的语法学起的，就像学习英语通常是从单词学起一样，然后使用单词构成句子，最后构成文章。本章我们将要学习的是 PHP 的基本语法，它的重要性不亚于英语单词的学习，是绝对的基础知识，读者应该将这部分知识牢固掌握。

2.1　PHP 的数据类型

所有存储在计算机中的内容都是数据。根据数据的大小，我们就可以给它合理分配一个空间以使得计算机资源得到合理利用，这在一些强类型语言中表现得淋漓尽致。我们学习的 PHP 是弱类型的语言，也就是对数据类型的敏感度较低，通常分配资源的过程都由语言系统自己完成，因此在学习中还是要轻松许多。

2.1.1　整型

整型就是不包含小数部分的数值，类似于我们在数学中学习的整数。编程语言中使用整型通常有正负和进制之分。进制就是进位制，PHP 中常用的有八进制、十进制和十六进制。下面来做一个简单的介绍。

1. 八进制整型

八进制整型数的特点是逢八进一，由数字 0～7 组成。最明显的特点就是八进制数中不会出现大于 8 的数值位。八进制整型的特点是：

- 八进制数值以 0 开始；
- 正八进制数值数值可以省略“+”号。

八进制数值的示例如表 2.1 所示。

表 2.1　八进制数值示例

+01367	正的八进制数值
-01367	负的八进制数值
01367	正的八进制数值
1367	错误的八进制数值，因为没有以 0 作为开始
2468	错误的八进制数值，因为出现了大于 8 的数值位

2. 十进制整型

十进制整型就是在生活中常用的进制，由数字 0～10 组成，使用逢十进一的进位制。

它的特点是正数可以省略“+”号。十进制数值的示例如表 2.2 所示。

表 2.2　十进制数值示例

+123789	正十进制数值
-123789	负十进制数值
10	正十进制数值

3. 十六进制整型

十六进制整型使用的进制是逢十六进一，由数字 0～9 和字母 A～F 组成。十六进制的特点是：

- ❑ 必须以 0x 或者 0X 作为开头；
- ❑ 正的十六进制数值可以省略“+”号；
- ❑ 构成的字母不区分大小写；

组成字母对应的十进制数值如表 2.3 所示。

表 2.3　十六进制字母对应十进制数值

十六进制字母	十进制数值	十六进制字母	十进制数值
A 或者 a	10	D 或者 d	13
B 或者 b	11	E 或者 e	14
C 或者 c	12	F 或者 f	15

十六进制数值示例如表 2.4 所示。

表 2.4　十六进制数值示例

+0x1237	正十六进制数值
-0x1237	负十六进制数值
0x1237	正十六进制数值
0x1ABD	正十六进制数值
0X2aBd	正十六进制数值
0x9f8G	错误的十六进制数值，因为有错误的数值位 G

2.1.2　浮点型

浮点型数值就是带有小数位的数值类型，由整数位、小数位和小数点（.）组成，整数位和小数位都由数字 0～9 组成。除了有整型的特性之外，浮点型有两种表示的方法。

- ❑ 十进制形式，如表 2.5 所示。
- ❑ 科学计数形式：该形式的特点是整数位通常用 1～10 之间的数值表示，如表 2.6 所示。

表 2.5　十进制形式浮点型

10.235	十进制形式
123.456	十进制形式
0.000456	十进制形式

表 2.6　科学计数形式浮点型

1.0235E1	等价于10.235
1.23456e2	等价于123.456
4.56E-4	等价于0.000456

2.1.3　字符型

字符型数据是不具有计算能力的文字，它包括中文字符、英文字符和数字字符等文字。字符型数据表示比较简单，如表 2.7 所示。

由多个字符组成的集合称为字符串，如表 2.8 所示。

表 2.7　字符型示例

壹	中文字符
1	数字字符
A	英文字符
#	特殊字符

表 2.8　字符串示例

Hello!	英文字符串
你好！	中文字符串
1234	数字字符串
@#￥%	特殊字符串

2.1.4　其他数据类型

除了上面我们介绍的一些常用的数据类型之外，PHP 中还有一些其他的数据类型，包括空类型、对象类型、资源类型和数组类型。

1. 空类型

空类型只有一个取值 NULL，用来表示没有任何数据。

2. 布尔型

布尔型只有两个取值 TRUE 和 FALSE，以下值被认为是 FALSE：

- 布尔值 FALSE 自身；
- 整型值 0；
- 浮点型值 0.0；
- 空字符串，以及字符串"0"；
- 不包括任何元素的数组；
- 特殊类型 NULL（包括尚未设定的变量）；
- 从没有任何标记（tags）的 XML 文档生成的 SimpleXML 对象。

所有其他值都被认为是 TRUE（包括任何资源）。其他的数据类型我们将在后续的学习中逐步为大家介绍。

2.2　变量和常量

前面我们已经学习了 PHP 的常用数据类型，计算机最基本的功能就是处理数据。我们可以把要处理的数据看做一个数据量，变量和常量的作用就是为这些数据量命名的。

2.2.1　变量

变量表示这个数据量可以被改变，该量在程序运行的不同时刻可能是不同的数据。

1. 变量名的命名规范

变量名就是为变量指定的名称，它需遵循如下规范：

- ❑ 变量名可以由大小写字母、数字和下划线组成；
- ❑ 变量名不可以数字作为开头；
- ❑ 为避免程序运行混乱，PHP 规定不可使用关键字作为变量名。PHP 常用的关键字如表 2.9 所示。

表 2.9　PHP 常用关键字

__LINE__	array	as	catch
throw	abstract	protected	and
or	xor	__FILE__	exception
break	case	class	const
continue	clone	try	this
final	php_user_filter	declare	default
die	do	echo	else
elseif	empty	enddeclare	endfor
endforeach	endif	endswitch	endwhile
eval	exit	extends	for
foreach	function	global	if
include	include_once	isset	interface
implements	extends	public	private
list	new	print	require
require_once	return	static	switch
unset	use	var	while
__FUNCTION__	__CLASS__	__METHOD__	

变量名的示例如表 2.10 所示。

表 2.10　变量名示例

abc	合法的变量名
_abc	合法的变量名
ABC123	合法的变量名
A_b_12	合法的变量名
123ABC	非法的变量名，因为是以数字开头的
return	非法的变量名，因为使用了 PHP 的关键字
_return	合法的变量名
@#￥%	非法的变量名，因为使用了下划线之外的特殊字符

2. 变量名的定义

将变量名与数据量建立关系的过程就称为变量的定义。PHP 中变量定义是用$（美元符）完成的，定义方法为在$后加一个符合 PHP 命名规范的变量名，即为数据量定义了一

个合法的变量名。

定义变量名示例如表 2.11 所示。

表 2.11　定义变量名示例

$_abc	合法的变量
$123	非法的变量，因为变量名是以数字作为开头
$return	非法的变量，因为使用关键字作为变量名

3. 变量的初始化

变量在定义以后，为了使用变量的过程中产生不可预料的结果，通常需要对其进行初始化。初始化变量就是为这个变量赋一个初始值，这个操作使用赋值符号“=”来完成，如下所示。

```
$a=123                    将变量 a 初始化为数值 123
$b="abc"                  将变量 b 初始化为字符串 abc
$c=NULL                   将变量初始化为 NULL
```

4. 变量的使用

在使用变量的时候需要同$符号一起使用，而不能直接使用变量名。

【示例 2-1】以下代码演示变量的使用形式。

```
<?php
    $a=123;              //变量初始化
    $b='abc';
    $c=NULL;
    echo $a;             //输出变量值
    echo $b;
    echo $c;
?>
```

运行结果如图 2.1 所示。

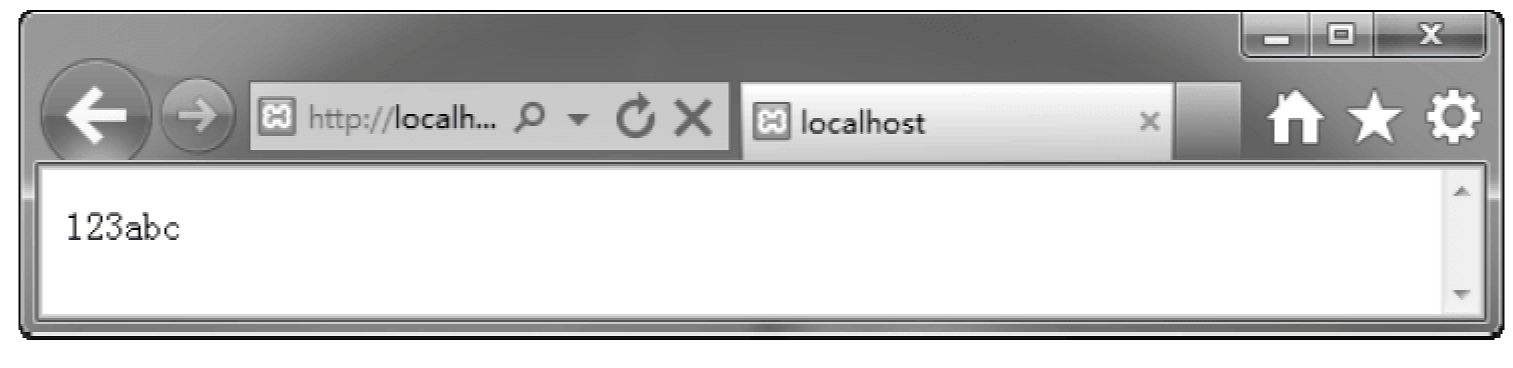


图 2.1　运行结果

在以上的代码中，由于$c 的值为 NULL，因此不会输出任何数据。

2.2.2　常量

常量就是在程序执行过程中不可以改变的量。常量在执行过程中始终都是一个指定的值，任何试图改变常量值的操作都是非法的。

1. 定义常量

常量的定义类似于变量的初始化，即常量在定义的同时就必须初始化。定义常量有两

种方式：

（1）PHP 5.3.0 之前的版本中使用 define()函数定义一个常量，它的语法如下：

```
bool define ( string $name , mixed $value [, bool $case_insensitive = false ] )
```

注意：由于在后面的学习中会多次使用如上的形式来说明一个函数的语法，它遵循的是一个固定的格式：

返回值　函数名（参数类型　参数 1，参数类型　参数 2，参数类型　参数 n…）

在中括号（[ ]）中定义的参数是可选参数，通常参数会说明默认值即 = 号后面的值。

参数 name 表示定义的常量名；参数 value 表示常量的值；参数 case_insensitive 用于设置常量对大小写是否敏感，默认 false 即为对大小写敏感。

【示例 2-2】以下代码演示使用 define()函数定义常量并输出其值。

```
<?php
    define('NUM',123);               //定义常量 NUM
    echo NUM;                        //输出 NUM 的值
    define('STR','ABC',TRUE);        //定义常量 STR 并设置大小写不敏感
    echo str;                        //输出常量 STR 的值，注意这里使用的是小写
?>
```

代码运行结果如图 2.2 所示。

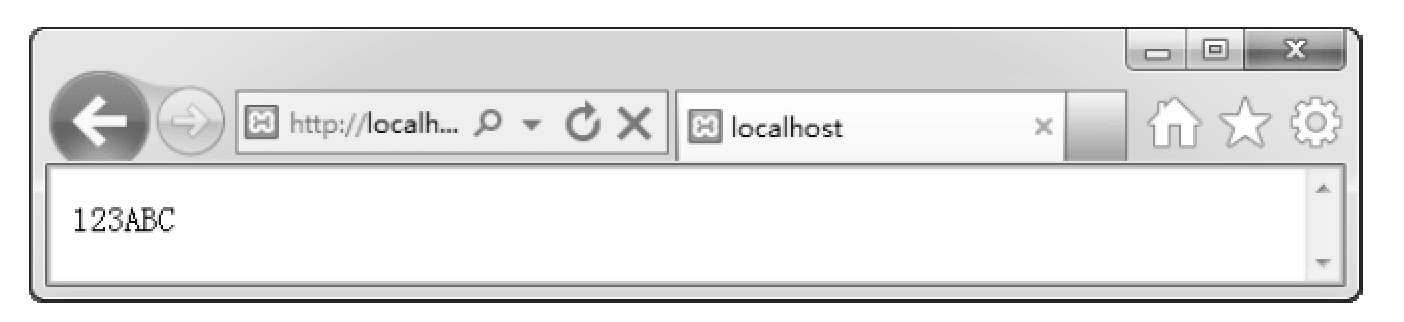


图 2.2　运行结果

从运行的结果中可以看到正确输出了定义的常量值。

（2）在 PHP 5.3.0 之后的版本中新增了使用赋值方式定义常量，语法如下：

```
const name = value
```

其中的 name 表示常量的名称；参数 value 表示常量的值。

【示例 2-3】以下代码演示使用赋值形式定义常量。

```
<?php
    const SUM=123;          //定义常量
    echo SUM;               //输出常量的值
    echo sum;               //这里会输出字符串 sum 而不是常量 SUM 的值
?>
```

代码运行结果如图 2.3 所示。

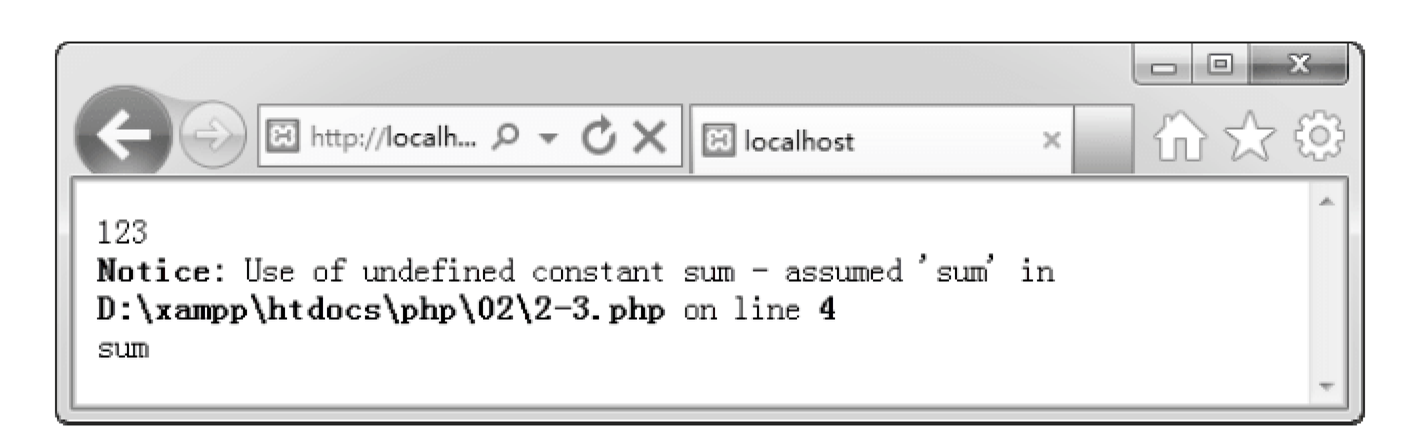


图 2.3　运行结果

从代码的运行结果可以看出，使用不一致常量名称不会输出期望的结果。

常量的两种定义方法虽然现在都可以正常使用，但是很有可能在后续的版本中不再支持 define()函数定义的方式。因此为了以后保持代码的兼容性，推荐使用最新的赋值方式定义常量。

2. 预定义常量

预定义常量就是系统中已经为我们定义好的常量，在使用的时候可以直接使用而不需要定义。常用的预定义常量如表 2.12 所示。

表 2.12　常用预定义常量

常 量 名	作 用 说 明
__FILE__	返回当前文件的名称（注意下划线都是两个）
__LINE__	返回当前代码所在的行号（注意下划线都是两个）
__FUNCTION__	返回所在函数的函数名（注意下划线都是两个）
__CLASS__	返回所在类的类名（注意下划线都是两个）
PHP_OS	返回操作系统的名称
PHP_VERSION	返回当前 PHP 服务器的版本
TRUE	代表布尔值，真
FALSE	代表布尔值，假
NULL	代表空值
M_PI	数学中的 π

【示例 2-4】以下代码演示输出当前文件的名称（使用__FILE__）。

```
<?php
    echo __FILE__;        //利用常量__FILE__输出当前文件的名称
?>
```

代码运行结果如图 2.4 所示。

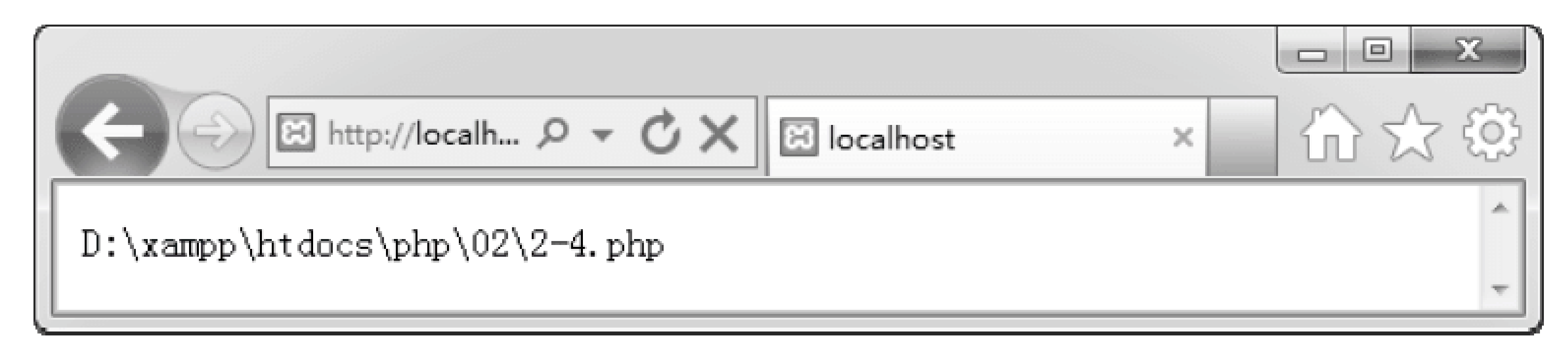


图 2.4　运行结果

以上代码的运行结果就显示了当前代码所在文件的路径及名称。

2.3　常用运算符

运算符是编程语言中不可或缺的一部分，本节我们将介绍一些 PHP 中常用的运算符，其中主要包括赋值运算符、算术运算符和连接运算符。运算符操作的数据称为操作数。

2.3.1　赋值运算符

赋值运算符是最基本的运算符，它用于为一个变量赋值或者为一个常量初始化。常量

的初始化在前面的小节中我们已经了解过了，本节就讲解为一个变量赋值的知识。为一个变量赋值有两种方式，即传值赋值和引用赋值。

1. 传值赋值

传值赋值是最常用的赋值方式，它用于将一个常量值赋值给一个变量或者将一个变量值的复制赋值给一个变量。使用的形式如下：

```
variable = constant                    //将常量赋值给变量
variable = variable                    //将变量的复制赋值给变量
```

由于将变量值赋值给变量是以变量值复制的形式进行，因此赋值变量和接受赋值的变量是两个不相关的变量，改变任意一个变量都不会影响到另外一个变量。

【示例 2-5】以下代码演示赋值运算符的传值赋值方式。

```
<?php
    $x=15;                  //将常量赋值给变量
    $y=$x;                  //将变量赋值给常量
    echo $x;                //输出变量 x 的值
    echo $y;                //输出变量 y 的值
    $y=25;                  //为变量 y 赋值
    echo $x;                //输出变量 x 的值
    echo $y;                //输出变量 y 的值
?>
```

代码运行结果如图 2.5 所示。

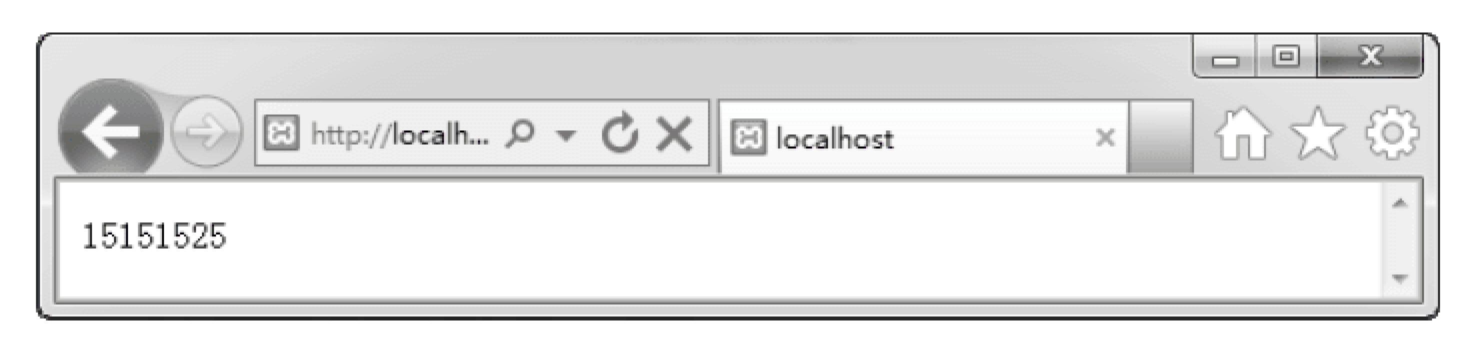


图 2.5　运行结果

从以上运行结果可以看出，变量 x 的值始终为 15 而没有随着变量 y 的改变而改变。

2. 引用赋值和取地址符

引用赋值相当于将变量中存储的量赋值给了一个变量，这时候两个变量名的都称为了该数据量的名称，因此操作任意一个变量的值都会对另一个变量产生影响。引用赋值需要使用到一个新的符号&（取地址符），它的用法如下：

```
variable = &variable            //引用赋值
```

【示例 2-6】以下代码演示赋值运算符的引用赋值方式。

```
<?php
    $x=16;              //初始化变量 x
    $y=&$x;             //引用赋值变量 y
    $z=&$y;             //引用赋值变量 z
    echo $x;            //输出变量 x 的值
    echo $y;            //输出变量 y 的值
    echo $z;            //输出变量 z 的值
```

```
        $y=26;              //为变量 y 赋值
        echo $x;            //输出变量 x 的值
        echo $y;            //输出变量 y 的值
        echo $z;            //输出变量 z 的值
        $z=36;              //为变量 z 赋值
        echo $x;            //输出变量 x 的值
        echo $y;            //输出变量 y 的值
        echo $z;            //输出变量 z 的值
    ?>
```

代码运行结果如图 2.6 所示。

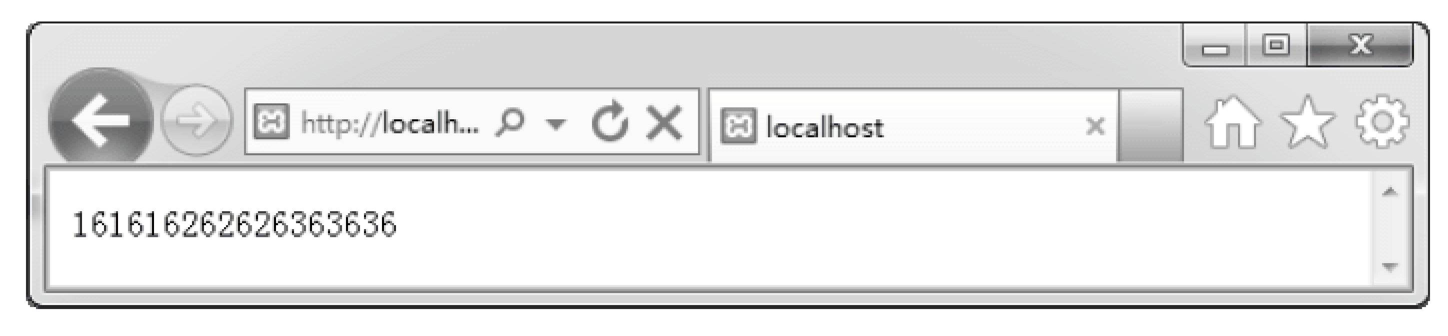


图 2.6　运行结果

从以上运行结果可以很明确地看出，改变其中任何一个变量的值都会引起其他变量的改变。

2.3.2　算术运算符

算术运算符用于对数值类型的变量或者常量进行算术运算，下面就来介绍这些运算符。

1. 取反运算符

取反运算符用于取得一个值的相反值，使用的符号为-（负号）。取反运算符的使用和理解比较简单，因此这里不做详细介绍。

2. 四则运算符

PHP 中的算数运算符同数学非常类似，最常用的例如+（加法运算符）、-（减法运算符）、*（乘法运算符）、/（除法运算符）运算符的使用方法同数字中的四则运算符相同。

【示例 2-7】以下代码演示使用四则运算符进行数值运算。

```
    <?php
        $x=10+15;               //加法运算
        $y=50-$x;               //减法运算
        $z=$y*2;                //乘法运算
        $n=$z/5;                //除法运算
        echo $x;                //输出变量 x 的值
        echo $y;                //输出变量 y 的值
        echo $z;                //输出变量 z 的值
        echo $n;                //输出变量 n 的值
    ?>
```

代码运行结果如图 2.7 所示。

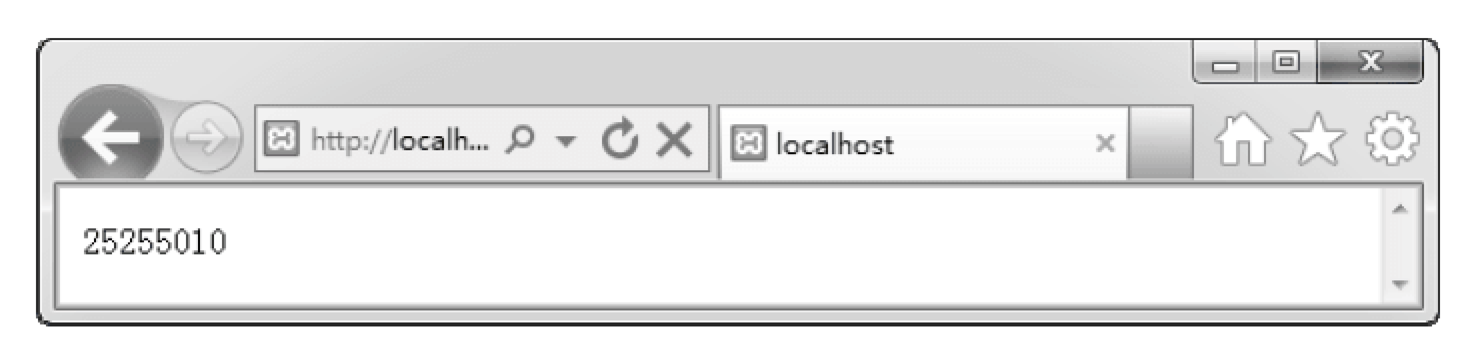


图 2.7　运行结果

以上演示了四则数学运算符的使用方法，同时输出对应的运算结果。

3. 取模运算符

取模运算符是编程语言中特有的一个运算符。执行取模运算的符号是“%”，它的作用是求出两个整型数值进行除法运算的余数。

【示例 2-8】以下代码演示求余运算符的使用。

```
<?php
	$x=10%5;			//进行取模运算
	$y=10%3;			//进行取模运算
	$z=10%6;			//进行取模运算
	echo $x;			//输出变量 x 的值
	echo $y;			//输出变量 y 的值
	echo $z;			//输出变量 z 的值
?>
```

代码运行结果如图 2.8 所示。

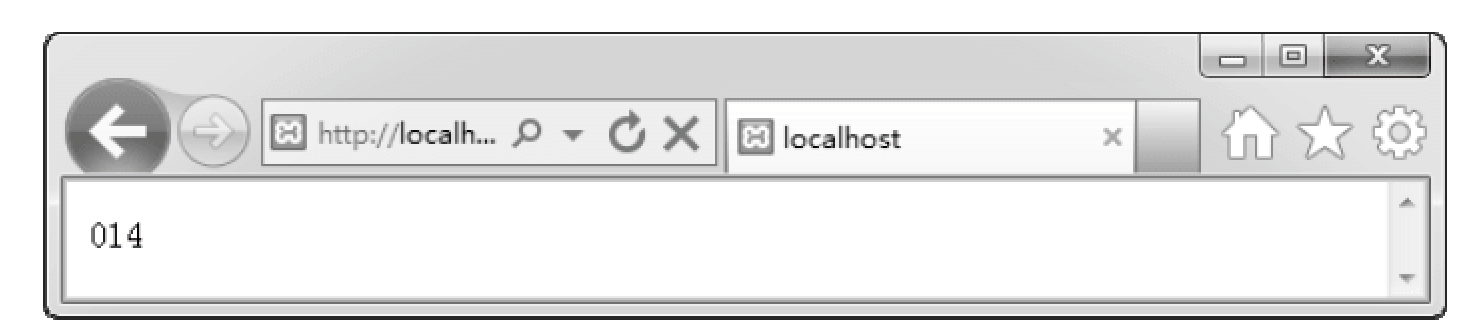


图 2.8　运行结果

以上输出结果就是代码中相应取模运算的结果。

4. 复合赋值运算符

复合赋值运算符是一类赋值运算的简化写法，我们在讲解之前先来看一个示例。

【示例 2-9】以下代码演示使用一类特殊的赋值用法。

```
<?php
	$x=10;				//初始化变量 x
	echo $x;			//输出变量 x 的值
	$x=$x+5;			//将变量 x 原来的值加 5 后赋值给自身
	echo $x;			//输出变量 x 的值
	$x=$x*5;			//将变量 x 原来的值乘以 5 后赋值给自身
	echo $x;			//输出变量 x 的值
	$x=$x/5;			//将变量 x 原来的值除以 5 后赋值给自身
	echo $x;			//输出变量 x 的值
	$x=$x-5;			//将变量 x 原来的值减 5 后赋值给自身
	echo $x;			//输出变量 x 的值
	$x=$x%7;			//将变量 x 原来的值与 5 取余后赋值给自身
	echo $x;			//输出变量 x 的值
?>
```

代码运行结果如图 2.9 所示。

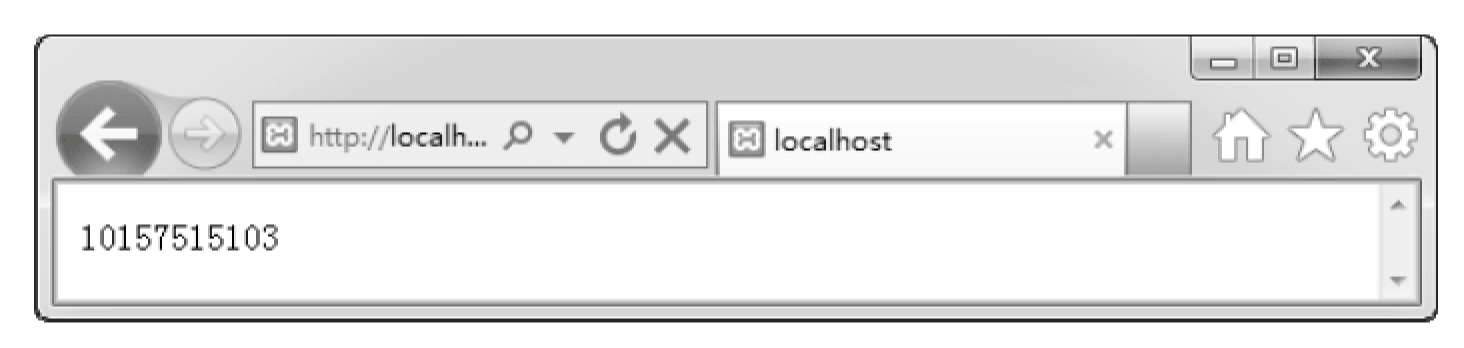


图 2.9　运行结果

上述的运算在传统的数学运算中是不会实现的，而在编程中这是可以实现的，我们也可以从运行结果中证明这类操作是不会出错的。这种运算的形式可以归纳为以下形式：

```
variable = variable operational other_variable
```

以上形式中的 operational 表示运算符；variable 表示一个变量；other_variable 表示不同于 variable 的其他变量。以上这种格式在编程中就可以简写为如下形式：

```
variable operational = other_variable
```

这种形式的运行效果是等价于上一种形式的，但是这绝不是为了美观或者使用简便，使用这种形式可以节省一定的资源，具体细节，读者可以参考其他相关资料，这里不做讲解。

【示例 2-10】以下代码使用复合赋值形式改写示例 2-9 中的代码。

```
<?php
    $x=10;              //初始化变量 x
    echo $x;            //输出变量 x 的值
    $x+=5;              //将变量 x 原来的值加 5 后赋值给自身
    echo $x;            //输出变量 x 的值
    $x*=5;              //将变量 x 原来的值乘以 5 后赋值给自身
    echo $x;            //输出变量 x 的值
    $x/=5;              //将变量 x 原来的值除以 5 后赋值给自身
    echo $x;            //输出变量 x 的值
    $x-=5;              //将变量 x 原来的值减 5 后赋值给自身
    echo $x;            //输出变量 x 的值
    $x%=7;              //将变量 x 原来的值与 5 取余后赋值给自身
    echo $x;            //输出变量 x 的值
?>
```

代码运行结果如图 2.10 所示。

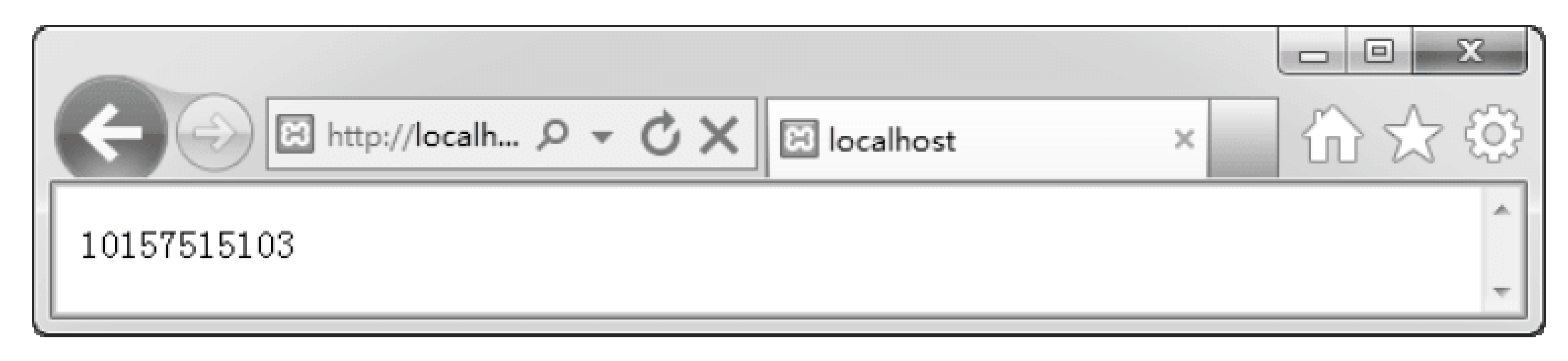


图 2.10　运行结果

从以上代码的运行结果可以看出，在使用复合赋值方式后计算结果与上一示例是相同的。

5. 递增递减运算符

递增递减运算符可以说是一种特定形式的复合赋值运算的简写，在讲解之前我们首先来看一个示例。

【示例 2-11】以下代码演示一类特殊的复合赋值运算。

```
<?php
    $x=1;               //初始化变量 x
    echo $x;            //输出变量 x 的值
    $x+=1;              //进行复合赋值操作
    echo $x;            //输出变量 x 的值
    $x+=1;              //进行复合赋值操作
    echo $x;            //输出变量 x 的值
```

```
    $x+=1;              //进行复合赋值操作
    echo $x;            //输出变量 x 的值
    $y=4;               //初始化变量 y
    echo $y;            //输出变量 y 的值
    $y-=1;              //进行复合赋值操作
    echo $y;            //输出变量 y 的值
    $y-=1;              //进行复合赋值操作
    echo $y;            //输出变量 y 的值
    $y-=1;              //进行复合赋值操作
    echo $y;            //输出变量 y 的值
?>
```

代码运行结果如图 2.11 所示。

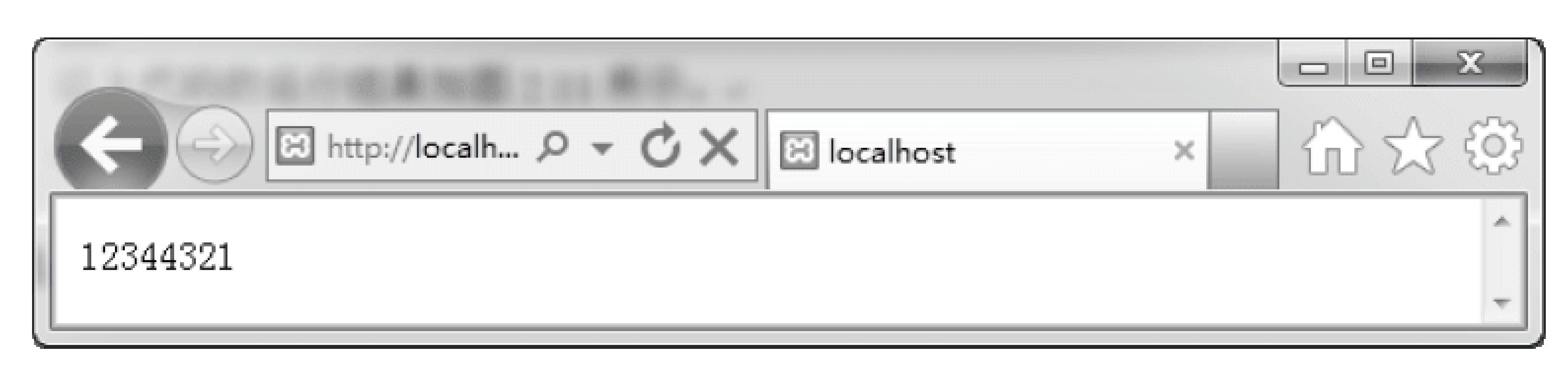


图 2.11　运行结果

以上代码的作用是对一个数做每次加 1 或者减 1 的操作，这种操作就称为递增或者递减运算，在循环操作中使用非常频繁。以上的操作同样可以归纳为一个固定的形式，如下所示。

```
variable (+ or -) = 1
```

以上这种递增或者递减的操作在编程语言中定义了两个对应的操作符++（递增运算符）和--（递减运算符）。因此，以上的形式就可以改写为如下形式：

```
variable++          //后置递增
variable--          //后置递减
++variable          //前置递增
--variable          //前置递减
```

从上面的形式中可以看出，递增和递减操作分为前置和后置：

- 前置递增或者递减运算的特点是，先进行递增或者递减运算然后再取变量的值；
- 后置递增或者递减运算的特点是，先进行对变量取值的操作，然后再进行递增或者递减的操作，因此这种操作的效果会在下一次使用该变量时体现。

【示例 2-12】以下代码演示递增运算的使用。

```
<?php
    $x=1;               //初始化变量 x
    $x++;               //后置递增变量 x
    echo $x;            //输出变量 x 的值，结果为 2
    echo $x++;          //输出变量 x++的结果。由于是后置递增，因此结果仍为 2
    echo $x++;          //输出变量 x++的结果。上条递增的结果在这里表现，因此此时
                          变量 x 的值为 3，因为此条运算为后置，因此结果为 3
    $y=1;               //初始化变量 y
    ++$y;               //进行前置递增运算
    echo $y;            //输出变量 y 前置递增后的值，结果为 2
    echo ++$y;          //输出变量 y 前置递增后的值。因为此时变量 y 的值为 2，
```

```
                        //因此前置递增后结果为 3
    echo ++$y;          //输出变量 y 前置递增后的值，结果为 4
?>
```

代码运行结果如图 2.12 所示。

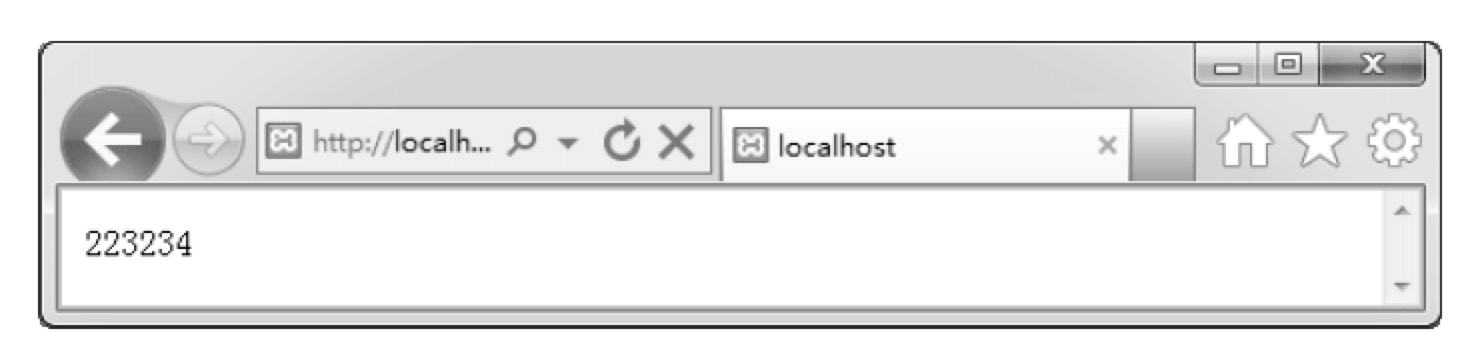


图 2.12　运行结果

以上代码演示的是前置和后置递增运算操作，递减操作的运行流程与递增操作类似，这里就不再做详细讲解。

2.3.3　连接运算符

连接运算符用于连接两个字符串，它使用的是符号“.（圆点）”，使用形式如下：

```
variable0.variable1.variable3.variablen
```

它也有复合的形式，如下所示。

```
variable.= (other_)variable
```

【示例 2-13】以下代码演示连接运算符的使用。

```
<?php
    $x=3;                   //初始化变量 x
    $y=5;                   //初始化变量 y
    $z=$x.$y;               //连接变量 x 和 y
    echo $z;                //输出变量 z 的值
    $m='abc';               //初始化变量 m
    $m.='xyz';              //使用复合形式连接字符串
    echo $m;                //输出变量 m 的值
    echo '<br />'.'Hello'.' world!';        //在输出时链接多个字符串
?>
```

代码运行结果如图 2.13 所示。

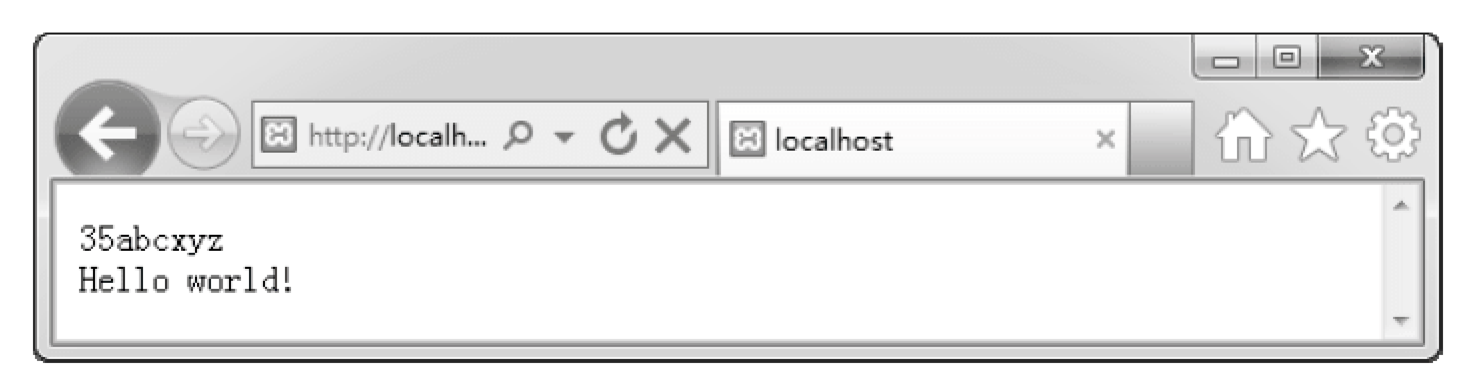


图 2.13　运行结果

这里需要注意的是连接操作符两侧的操作数均为字符型数据，若为其他类型则系统会将其转换为字符型。因此，以上运行结果中的 35 是字符串而不是数值。

2.3.4　比较运算符

比较运算符用来对两个值进行比较，比较结果成立则这个比较运算的结果即为 TRUE；

不成立则为 FALSE。比较运算符的符号和说明如表 2.13 所示。

表 2.13　比较运算符

符　号	名　称	说　　明
==	等于	两个操作数在忽略类型的情况下相等则为 TRUE
===	全等	两个操作数相等并且它们的类型也相同则为 TRUE
!=	不等	两个操作数在忽略类型的情况下相等则为 TRUE
<>	不等	不等的另一种表示方式
!==	非全等	两个操作数不相等或者类型不相同则为 TRUE
<	小于	做操作数小于右操作数则为 TRUE
>	大于	左操作数大于右操作数则为 TRUE
<=	小于等于	左操作数小于等于右操作数则为 TRUE
>=	大于等于	左操作数大于等于右操作数则为 TRUE

比较操作符理解和使用都比较简单，因此我们只通过一个简单的示例来做讲解即可。

【示例 2-14】以下代码演示比较运算符的使用。

```
<?php
    $a=5;                        //将变量 a 初始化为数值 5
    echo '$a='.$a;
    $b=6;                        //将变量 b 初始化为数值 6
    echo '<br />$b='.$b;
    $c='5';                      //将变量 c 初始化为字符串 5
    echo '<br />$c='.$c;
    $d=6;                        //将变量 d 初始化为数值 6
    echo '<br />$d='.$d;
    //进行比较运算并输出计算结果
    echo '<br />$a>$b:'.($a>$b);
    echo '<br />$a<$b:'.($a<$b);
    echo '<br />$a==$c:'.($a==$c);
    echo '<br />$b==$d:'.($b==$d);
    echo '<br />$a===$c:'.($a===$c);        //数值类型与字符串型不全等，因此输出为 FALSE
    echo '<br />$b===$d:'.($b===$d);
    echo '<br />$a!=$c:'.($a!=$c);
    echo '<br />$a!==$c:'.($a!==$c);
    echo '<br />$a<=$b:'.($a<=$b);
    echo '<br />$b>=$d:'.($b>=$d);
?>
```

代码运行结果如图 2.14 所示。

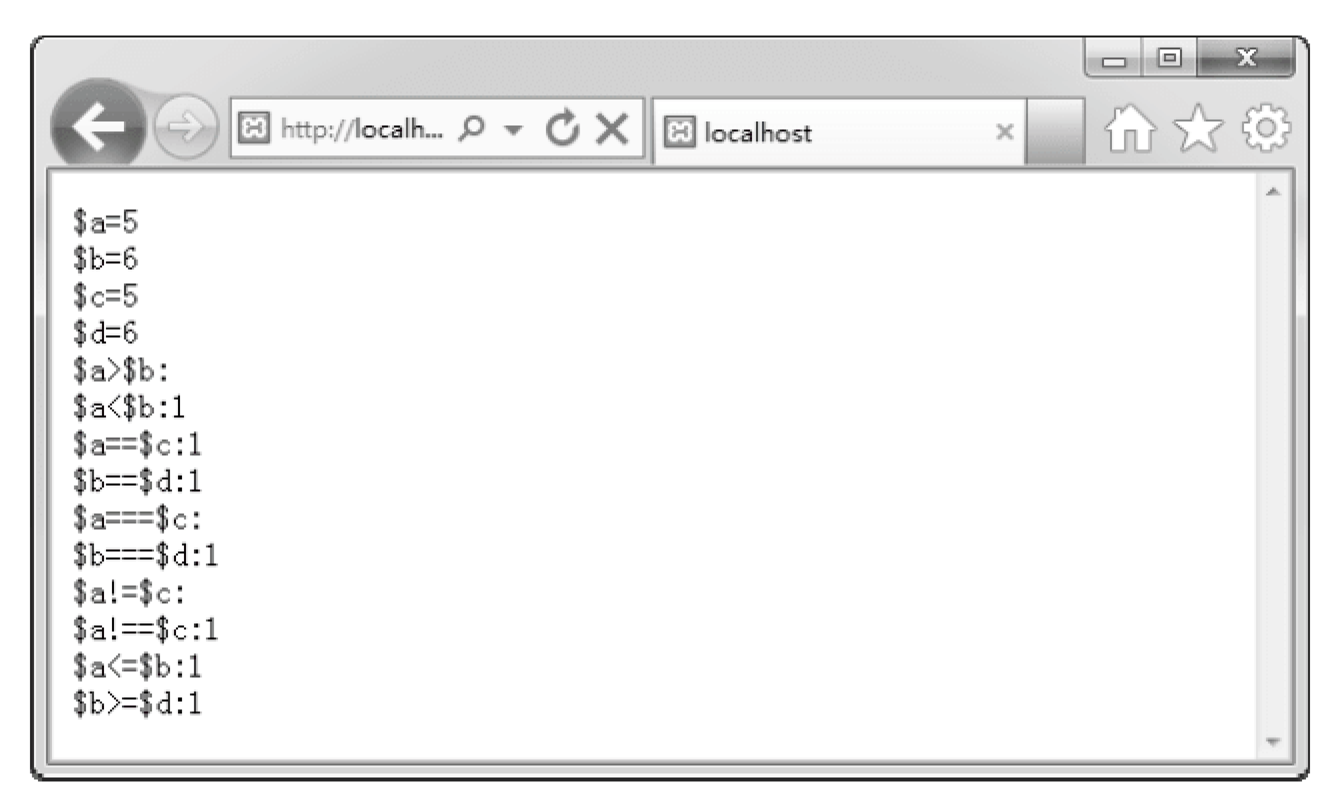


图 2.14　运行结果

这里我们需要明白的是，通常情况下输出 FALSE 在浏览器中就表现为空，即无任何输出。输出 TRUE 则通常表现为 1。因此读者在这里一定不要迷惑运行的输出结果。

2.3.5　逻辑运算符

逻辑运算被用来使用数学运算解决逻辑问题。我们可以这么理解这个概念，例如显示器要显示图像就需要显示器是完好的而且有图像输入源，这两个必须的条件如果满足就可以显示，不满足就不可以显示。逻辑运算符就是用来判断这两个条件是否满足的。逻辑运算符以及说明如表 2.14 所示。

表 2.14　逻辑运算符

符号	名　称	说　明
and	逻辑与	两个操作数均为 TRUE 则为 TRUE
or	逻辑或	两个操作数中至少一个为 TRUE 则为 TRUE
xor	逻辑异或	两个操作数有一个但不同时为 TRUE 则为 TRUE
!	逻辑非	操作数为 FALSE 则为 TRUE
&&	逻辑与	逻辑与的另一种表示方法但是优先级要高于 and
‖	逻辑或	逻辑或的另一种表示方法但是优先级要高于 or

【示例 2-15】以下代码演示逻辑运算符的使用。

```
<?php
    //输出各种逻辑运算的运行结果
    echo 'TRUE and FALSE:'.(TRUE and FALSE);
    echo '<br />TRUE or FALSE:'.(TRUE or FALSE);
    echo '<br />TRUE xor FALSE:'.(TRUE xor FALSE);
    echo '<br />!FALSE:'.(!FALSE);
    echo '<br />TRUE && FALSE:'.(TRUE && FALSE);
    echo '<br />TRUE || FALSE:'.(TRUE || FALSE);
?>
```

代码运行结果如图 2.15 所示。

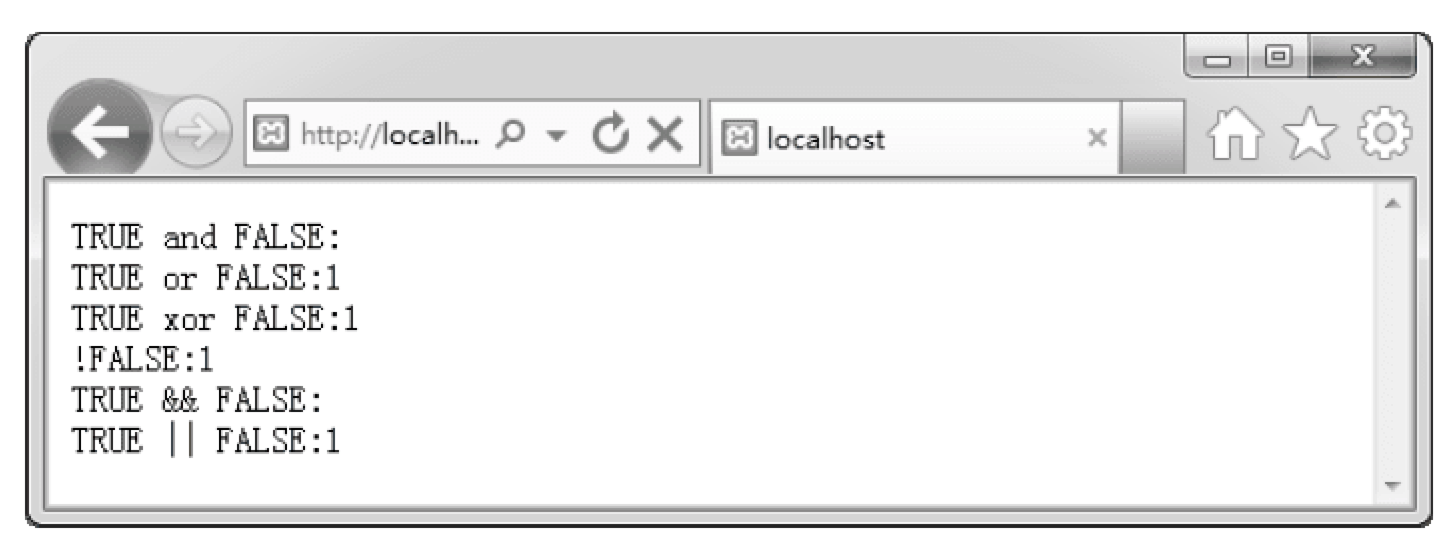


图 2.15　运行结果

读者可以根据以上的运行结果加深理解。

2.3.6　三元运算符

三元运算符是 PHP 中唯一一个可以操作三个操作数的运算符，它的语法形式如下：

```
(expr1) ? (expr2) : (expr3)
```

其中的 expr1 为一个布尔类型的表达式，如果 exp1 的值为 TRUE，则表达式的值为 expr2；为 FALSE，则表达式的值为 expr3。

【示例 2-16】以下代码演示三元运算符的使用。

```
<?php
    $a=5;            //初始化两个变量
    $b=15;
    $c=($a>$b)?$a:$b;
    echo '两个变量中比较大的数值是：'.$c;
    $x=TRUE;         //初始化两个变量
    $y=FALSE;
    $z=($x&&$y)?'显示器显示图像':'显示器不显示图像';
    echo '<br />'.$z;
?>
```

代码运行结果如图 2.16 所示。

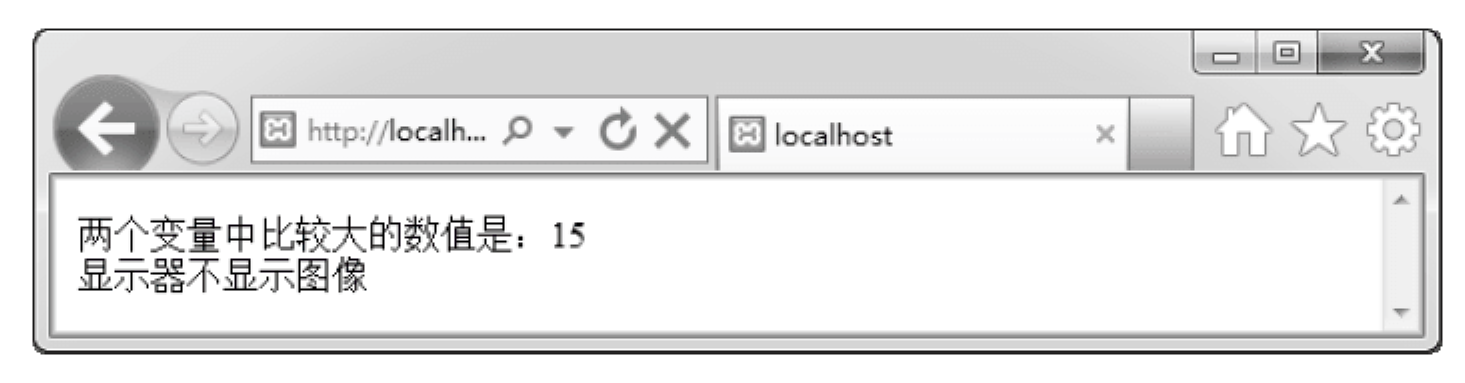


图 2.16　运行结果

2.3.7　其他运算符

除了以上我们学习的运算符之外，PHP 还有一些其他的运算符，包括错误控制运算符、执行运算符、数组运算符、类型运算符和位运算符。由于我们现在的知识还有欠缺，因此这些运算符将在以后知识达到一定高度的时候讲解。

2.3.8　运算符的优先级

运算符的优先级即用来指定在使用多个运算符的代码中首先执行哪种运算。运算符的优先级表如表 2.15 所示。

表 2.15　运算符优先级表

优 先 级	运　算　符
1	clone new
2	[
3	++ --
4	~ - (int) (float) (string) (array) (object) (bool) @
5	instanceof
6	!
7	* / %
8	+ - .
9	<< >>
10	< <= > >= <>
11	== != === !==

续表

<table>
<tr><th>优 先 级</th><th>运　算　符</th></tr>
<tr><td>12</td><td>&</td></tr>
<tr><td>13</td><td>^</td></tr>
<tr><td>14</td><td>|</td></tr>
<tr><td>15</td><td>&&</td></tr>
<tr><td>16</td><td>||</td></tr>
<tr><td>17</td><td>? :</td></tr>
<tr><td>18</td><td>= += -= *= /= .= %= &= |= ^= <<= >>=</td></tr>
<tr><td>19</td><td>and</td></tr>
<tr><td>20</td><td>xor</td></tr>
<tr><td>21</td><td>or</td></tr>
<tr><td>22</td><td>,</td></tr>
</table>

运算符按照优先级从高到低执行，在代码中可以使用小括号来改变优先级并且使得程序更加易读。

2.4　输出语句 echo

echo 我们在前面知识的学习过程中已经多次使用，这里再另外作为一节来讲解是因为它有一些不同的用法和技巧，需要我们学习和掌握。echo 准确地来说是一个语言结构，因此不一定要使用小括号来指明参数，单引号或双引号都可以。但是单引号和双引号的使用结果是不同的。

（1）单引号形式不会将其中的变量名解析，即类似于以下形式不会被解析：

```
$var_name
```

【示例 2-17】以下代码演示单引号形式的使用。

```
<?php
    $x=12;              //初始化变量 x
    echo '$x';          //单引号形式输出
?>
```

代码运行结果如图 2.17 所示。

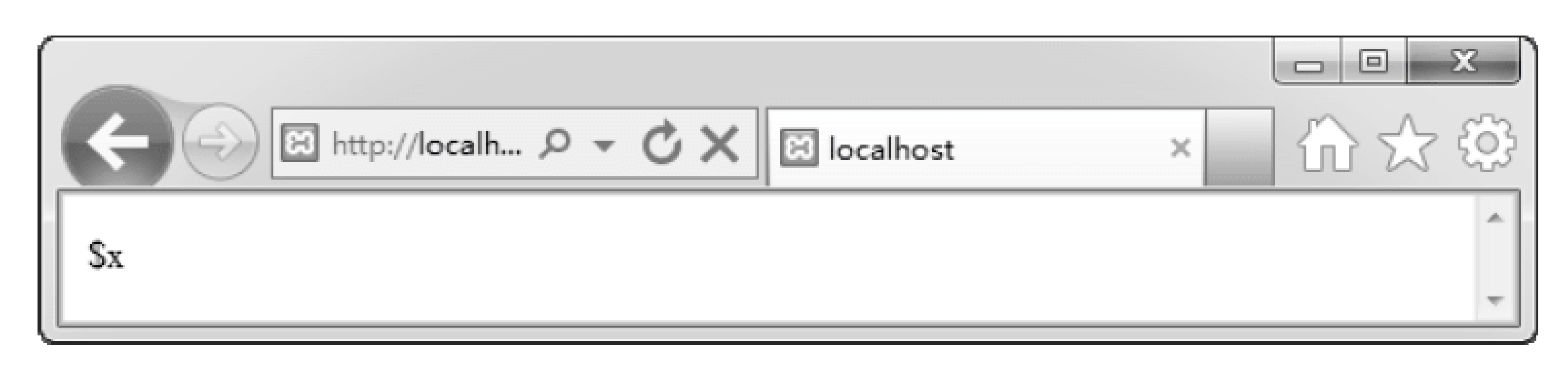


图 2.17　运行结果

以上运行结果是和我们前面所使用的无异，输出结果大家也应该是预料到了。

（2）双引号形式会将其中的变量名解析。

【示例 2-18】以下代码演示双引号形式的使用。

```
<?php
    $x=15;                //初始化变量 x
    echo "$x";            //双引号形式输出
?>
```

代码运行结果如图 2.18 所示。

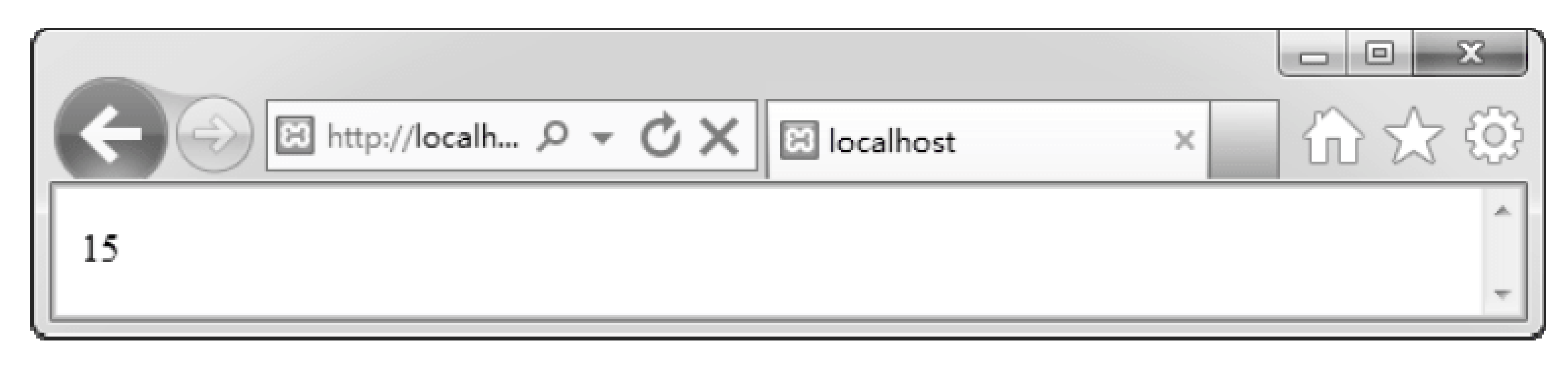


图 2.18　运行结果

这个运行结果可能与大部分读者预料的不同，其中的变量被解析后输出了变量的值。

在双引号形式中如果想得到同单引号形式相同的效果，我们就需要使用\（转义符）来完成，它可将一些特殊的字符转义进而正常输出。

【示例 2-19】以下代码演示转义符的使用。

```
<?php
    $x=15;                    //初始化变量 x
    echo "\$x";               //进行转义输出
    echo '<br />$x='.$x;      //单引号形式输出
    echo "<br />\$x=$x";      //双引号形式输出
?>
```

代码运行结果如图 2.19 所示。

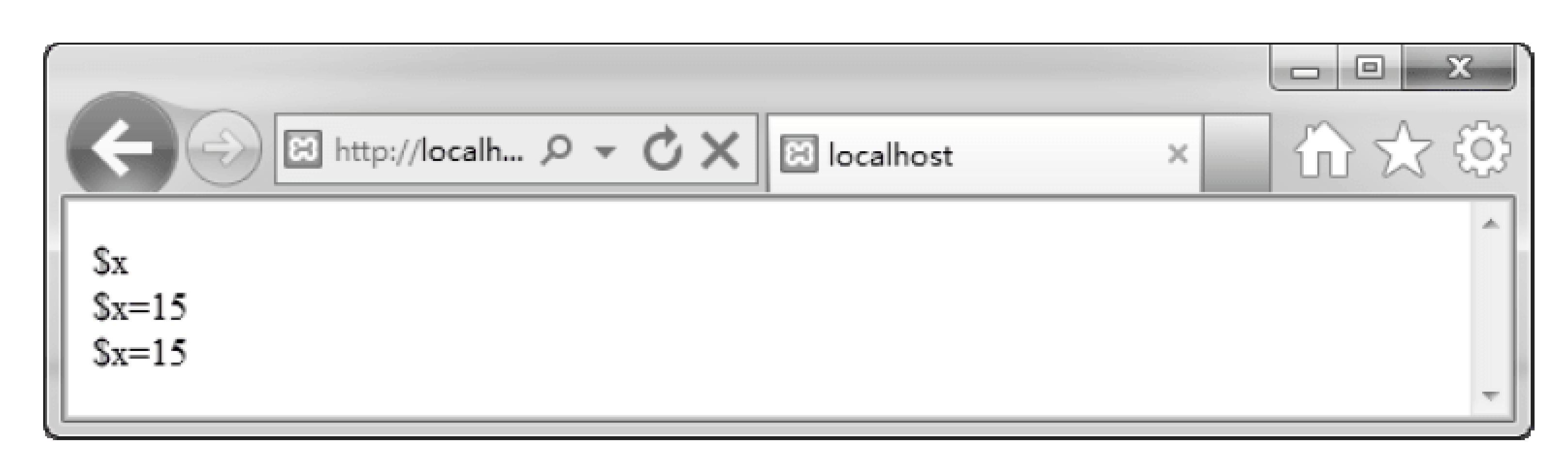


图 2.19　运行结果

以上代码中的最后两句输出，分别使用单引号形式和双引号形式输出了同样的结果，读者应该学会这种使用方法。

（3）echo 可以输出使用逗号分隔的多个参数。

【示例 2-20】以下代码演示 echo 输出多个参数。

```
<?php
    $x=5;                                //初始化两个变量
    $y=10;
    echo $x,$y,"<br />$x+$y=",$x+$y;     //输出多个参数
?>
```

代码运行结果如图 2.20 所示。

结合运行结果可以看出，使用 echo 一次输出了多个参数，而在没有学习这个知识以前通常是做不到的。

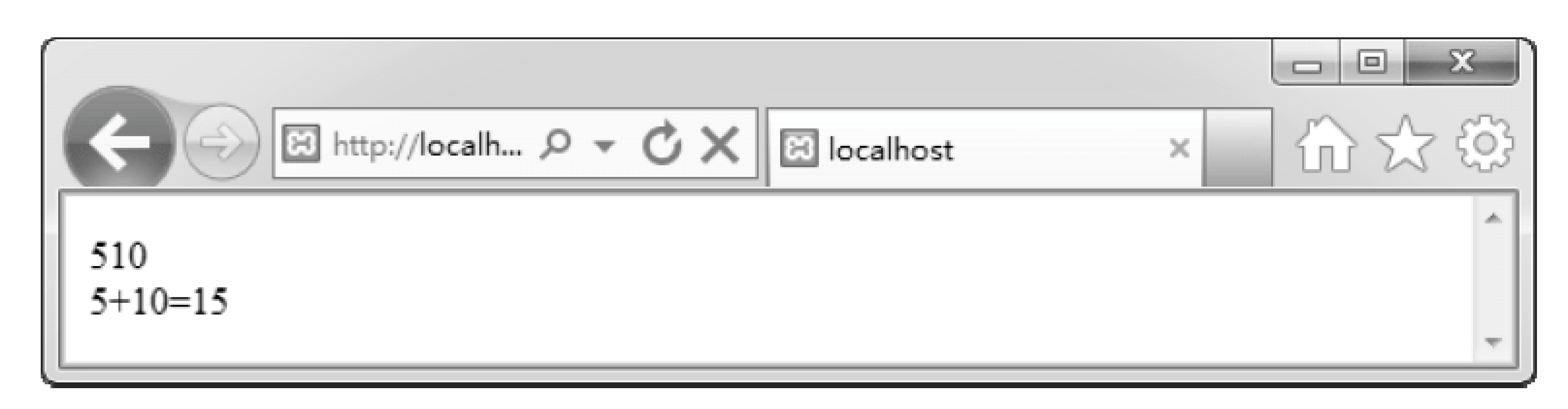


图 2.20 运行结果

2.5 小 结

本节主要讲解了 PHP 的数据类型和运算符的知识，虽然内容不多但是需要掌握的知识还是很多的。逻辑运算符和三元运算符对初学者来说是不太好理解的，读者应该多做练习，从实践中学到知识，为以后更好地学习做好基础。

2.6 本章习题

1. 将字符串“Hello”赋值给一个变量并在浏览器输出。
2. 将你的名字定义为一个常量并在浏览器输出。
3. 将习题 1 和 2 中定义的变量和常量连接起来输出。
4. 计算半径为 5 的圆面积和周长，并以图 2.21 所示的形式输出。

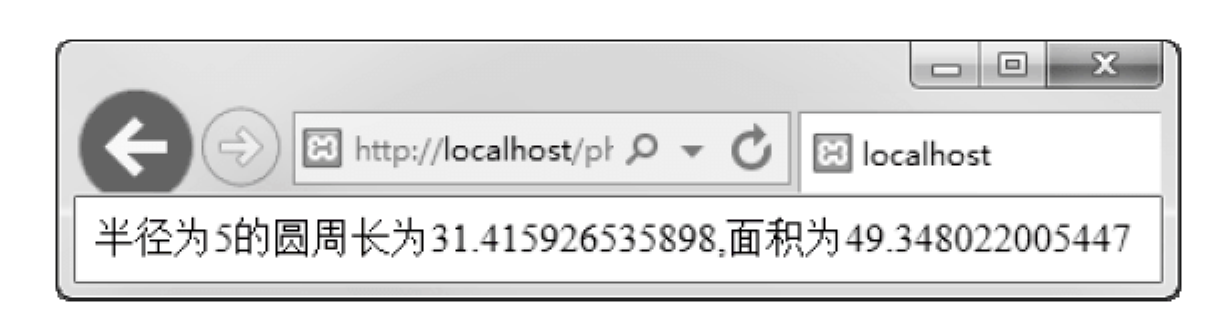


图 2.21 运行效果

5. 编写程序，计算表 2.16 中表达式的值，将表补充完整。

表 2.16 完成下表

表 达 式	值
2+3-5	
2*3-5	
2+3-5&&2*3-5	
2+3-5&&2*3-5?2+3-5:2*3-5	

第 3 章　语 言 结 构

上一章我们学习了 PHP 的数据类型和常用的运算符，这些知识在本节中将作为语言的基本组成部分。本章将要讲解的内容是语言结构的知识。语言结构是一个程序的整体框架，它用来主导程序的走向，是程序的核心。

3.1　语　　句

语句是组成一个程序的最基本组成部分。经过各种语句的协调工作，即可构成一个功能完整的程序。

3.1.1　表达式

表达式是指由常量和变量等通过运算符连接起来而形成的一个有意义的算式。在前面的章节中我们已经多次使用过，如下即为一些表达式：

```
$x=6
$y+5-$z
$x>$y?$x:$y
```

3.1.2　表达式语句

在一个有效表达式的结尾加入一个分号即为一个表达式语句。如下所示即为表达式语句：

```
$x=6;             //赋值语句
$x++;             //递增语句
$x>$y;            //判断语句
```

表达式语句和表达式的区别在于：表达式代表的是一个数值，而表达式语句代表的是一种动作特征。PHP 程序中最常见的表达式语句为赋值语句。

3.1.3　复合语句和空语句

复合语句是使用花括弧将多条语句组合而成的一种语句格式，也称为语句块。复合语句从形式上看是多个语句的组合，但在语法意义上是一个整体，被看作一条语句。所以只要是可以使用简单语句的地方都可以使用复合语句，如下所示即为复合语句：

```
{                      //左花括弧表示复合语句开始
    $x='Hello';
    $y='world';        //符合语句中的多条语句
```

```
    $z=$x+$y;
    echo $z;
}                    //右花括弧表示复合语句结束
```

空语句即为不执行任何操作的语句，它的形式如下：

```
;                    //空语句只有一个分号
```

空语句常用于在某些场合占据一个语句的位置，例如 for 循环之中。

3.1.4　语句的执行顺序

在没有控制结构的语句中，程序是由程序开头逐句执行直到没有语句为止，但是这种结构有些功能是完成不了的，例如根据不同的状态输出不同的信息的操作。要完成这类操作，就需要用到一些语言结构，这就是我们接下来要学习的知识。

3.2　选 择 语 句

选择语句用于使程序在不同的条件下执行不同的语句。PHP 中的选择语句有 if 语句和 switch 语句，下面就来介绍他们。

3.2.1　if 语句

if 语句也称为条件语句，它有多种使用形式，包括 if 形式、if…else 形式和 if…elseif…else 形式，下面分别介绍这些形式的使用。

1. if 形式

if 形式是 if 语句最基本的形式，它的语法结构如下：

```
if (表达式)
    语句 1;
```

如果表达式的值为 TRUE，则执行语句 1，如果为 FALSE 则执行语句 1 之后的语句，这里的语句 1 可以是一个语句，也可以是一个复合语句。如果是复合语句，必须带有花括号。该结构的流程如图 3.1 所示。

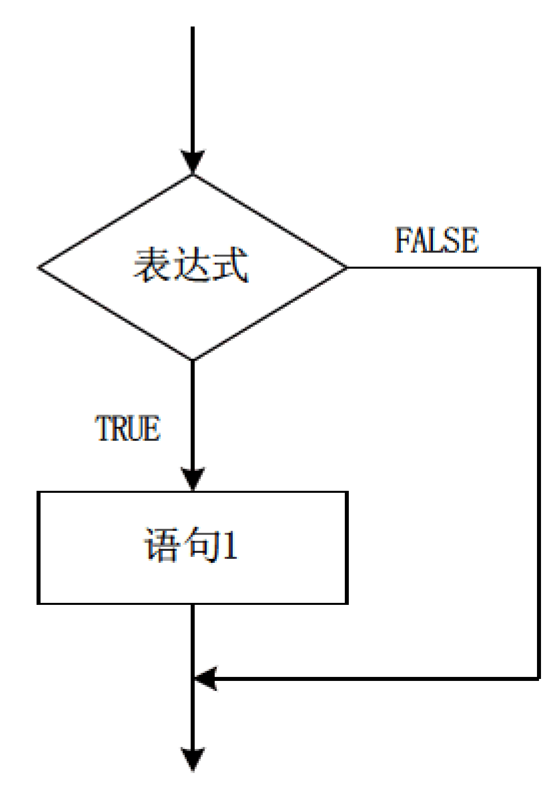


图 3.1　if 形式流程图

【示例 3-1】以下代码演示 if 选择语句的 if 形式用法。

```
<?php
    $x=6;                                   //初始化两个变量
    $y=7;
    if($y>$x)                               //比较运算结果为 TRUE
        $x=$y;                              //该语句将$y 的值赋值给$x
    echo "两个变量中比较大的值是$x 。";     //输出两个变量中比较大的值
?>
```

代码运行结果如图 3.2 所示。

以上代码的作用就是输出两个变量中比较大的值，如果变量 y 的值小于变量 x，则赋

值语句不会被执行。通过 if 语句来保证$x 值永远为最大的。

2. if…else 形式

if…else 形式的语法结构如下：

```
if (表达式)
    语句 1;
else
    语句 2;
```

如果表达式的值为 TRUE，则执行语句 1，为 FALSE 则执行语句 2，该结构的流程如图 3.3 所示。

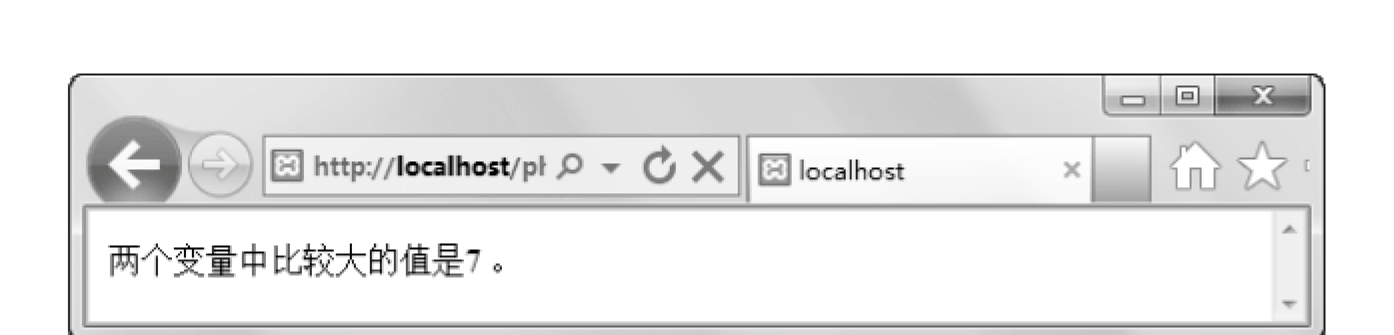


图 3.2　运行结果

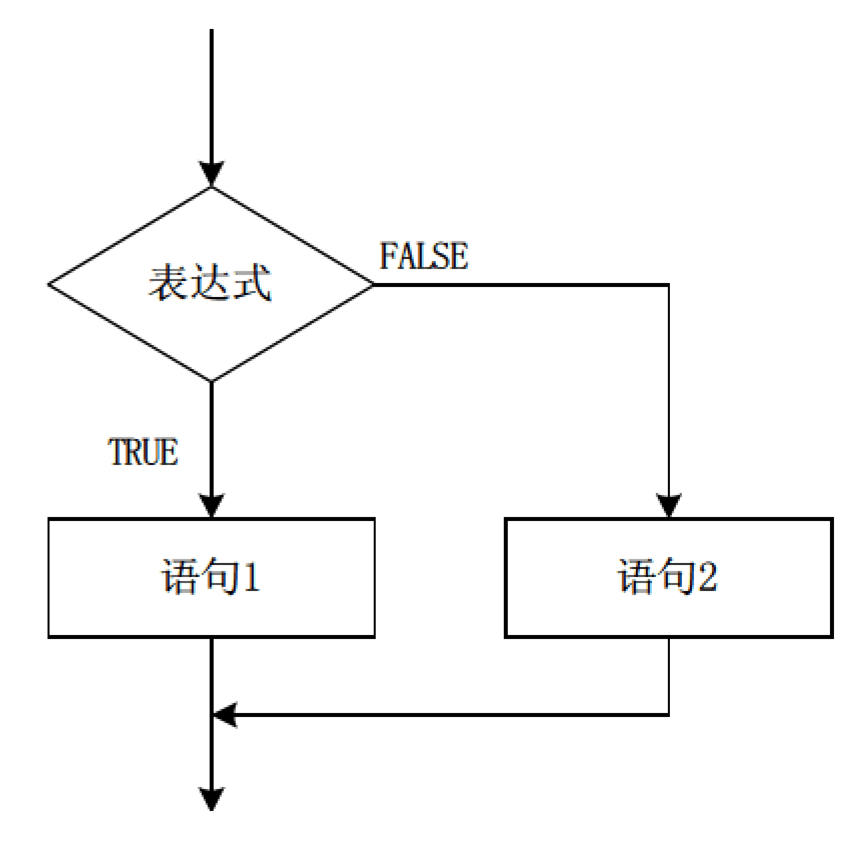


图 3.3　if…else 形式流程图

【示例 3-2】以下代码演示 if 选择语句的 if…else 形式用法。

```
<?php
    $x=16;                //初始化两个变量
    $y=17;
    if($x>$y)             //进行比较运算并根据比较结果执行对应的语句
        echo "两个变量中比较大的值是$x";
    else
        echo "两个变量中比较大的值是$y";
?>
```

代码运行结果如图 3.4 所示。

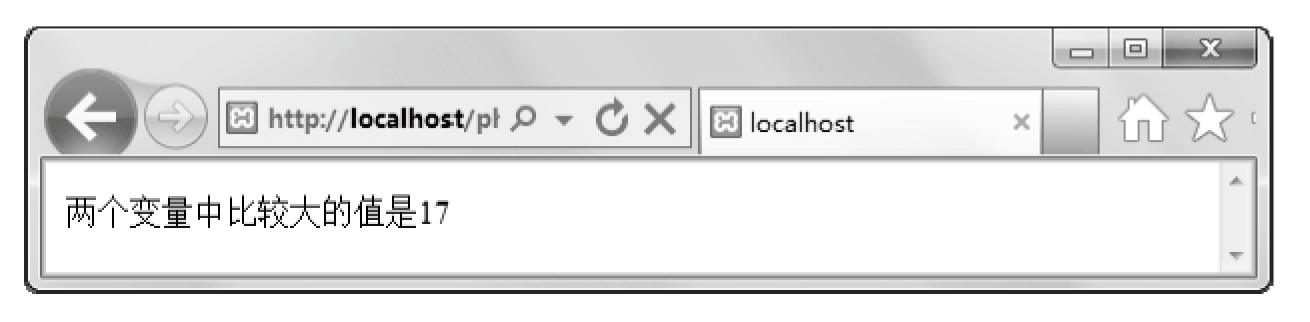


图 3.4　运行结果

由于变量 y 的值要大于变量 x 的值，因此 else 下的语句被执行。

3. if…elseif…else

if…elseif…else 的语法结构如下：

```
if (表达式 1)
```

```
    语句 1;
elseif(表达式 2)
    语句 2;
else
    语句 3;
```

🔔注意：以上语法结构中的 elseif 项可以有若干个，这里只列出了最基本的形式。该语法结构中的 else 从句是可选的。

这种结构从上到下逐个对条件进行判断，一旦条件满足或者遇到 else 从句就执行与相关的语句，并跳过结构中的其他代码。该结构的流程图如图 3.5 所示。

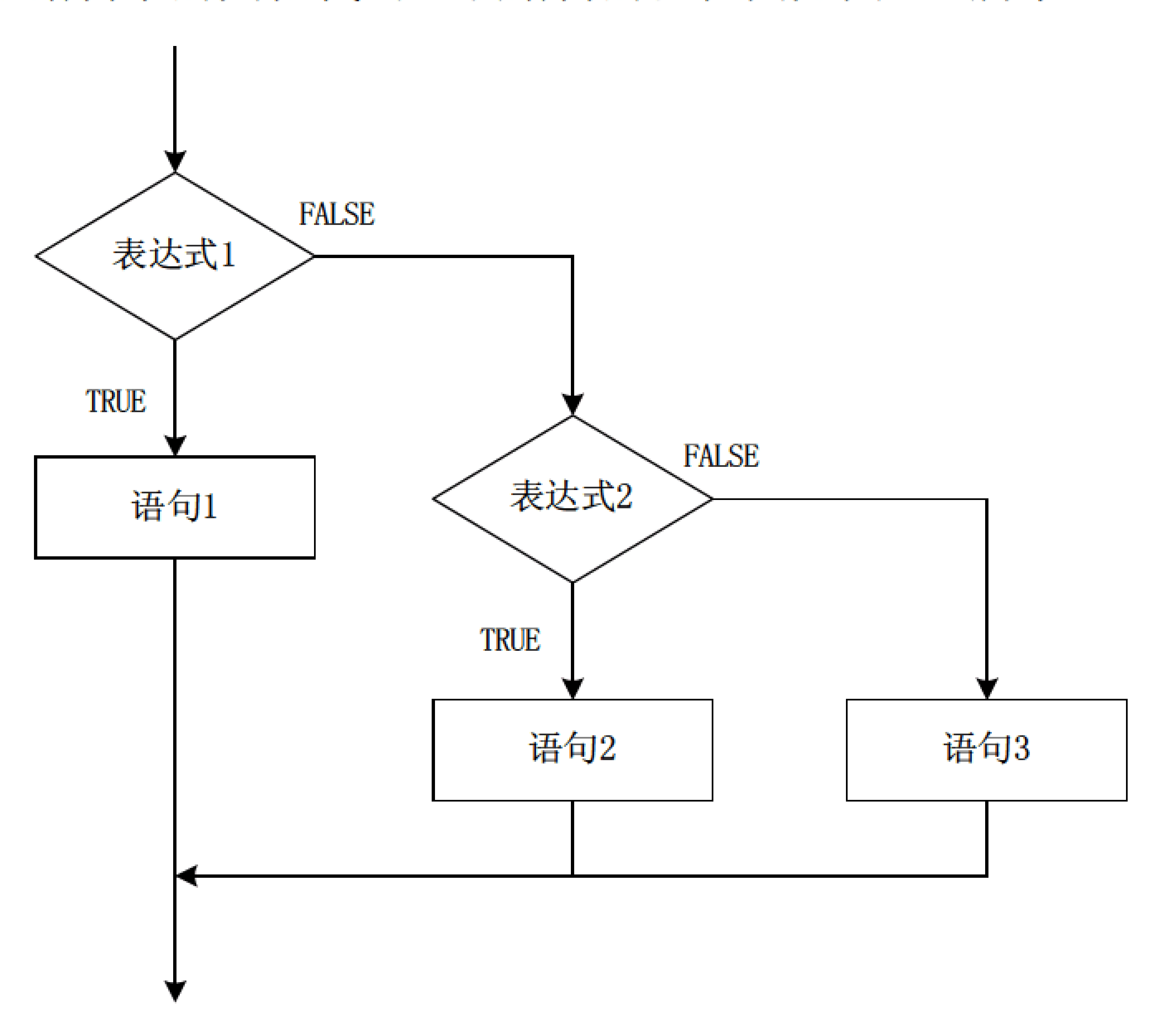


图 3.5　if…elseif…if 形式流程图

【示例 3-3】以下代码演示 if 选择语句的 if…elseif…else 形式的用法。

```
01  <?php
02      $score=90;                          //定义一个分数变量并初始化
03      if($score>=90&&$score<=100)         //对分数进行判断并输出对应评价
04          echo '这是一个优秀的成绩。';
05      elseif($score>=80&&$score<90)       //对分数进行判断并输出对应评价
06          echo '这是一个良好的成绩。';
07      elseif($score>=60&&$score<80)       //对分数进行判断并输出对应评价
08          echo '这个成绩需要努力。';
09      elseif($score<60&&$score>=0)        //对分数进行判断并输出对应评价
10          echo '这个成绩非常糟糕！';
11      else                        //不满足以上任何一个条件则输出合法性提示
12          echo '请确定成绩的合法性。';
13  ?>
```

当以上代码中$score=90 时，由于分数在 90～100 之间，因此第 4 行代码会被执行，输出结果如图 3.6 所示。

当$score=59 时，由于分数在 0～59 之间，因此第 10 行代码会被执行，输出结果如图 3.7 所示。

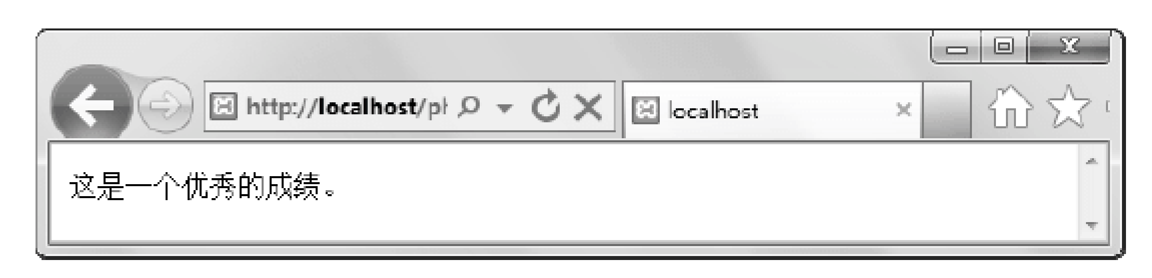


图 3.6　运行结果

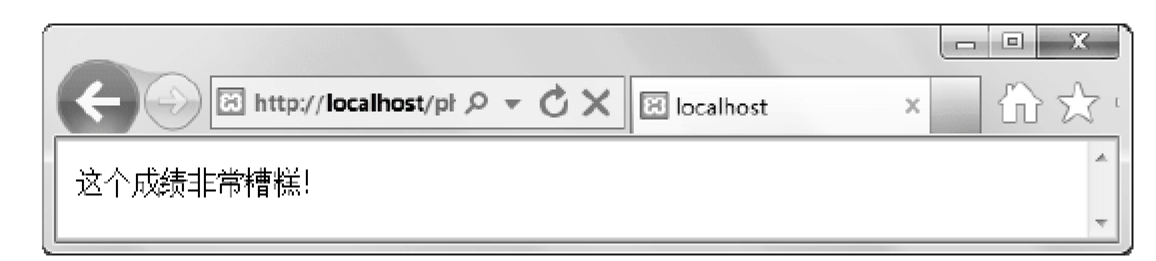


图 3.7　运行结果

当$score 为一个大于 100 或者小于 0 的数值时，default 后的语句会被执行，输出结果如图 3.8 所示。

4. if 语句嵌套

选择语句可以嵌套，也就是在一个选择结构中存在另一个选择结构，这是经常碰到的情况，但也是容易出错的地方，原因常出现在 if 和 else 的匹配问题。PHP 中的 else 总是会与最近的 if 匹配。我们首先来看一个使用正确嵌套的示例。

【示例 3-4】以下代码演示正确使用 if 语句嵌套。

```
<?php
    $operator='/';                  //定义一个变量并初始化
    $x=15;                          //定义两个操作数并初始化
    $y=0;
    if($operator=='+')              //判断运算符并执行相应运算
        echo "$x+$y=".$x+$y;
    elseif($operator=='-')
        echo "$x-$y=".$x-$y;
    elseif($operator=='*')
        echo "$x*$y=".$x*$y;
    elseif($operator=='/'){
        if($y==0)                   //循环嵌套，判断除法中的除数是否为 0
            echo '除数为 0，计算错误！';
        else
            echo "$x/$y=".$x/$y;
    }
    else
        echo '请输入一个正确的运算符！';
?>
```

代码运行结果如图 3.9 所示。



图 3.8　运行结果

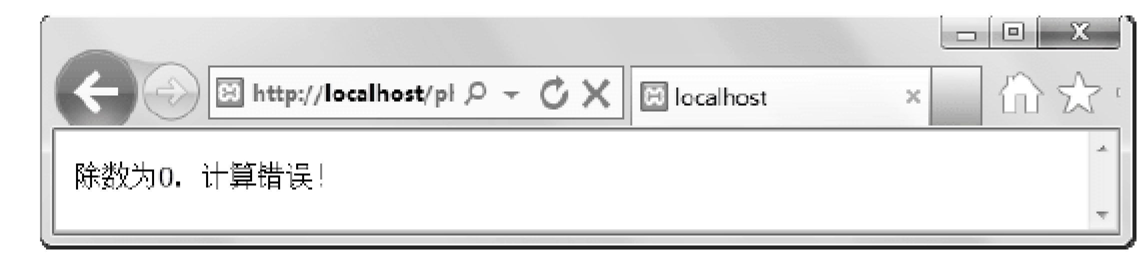


图 3.9　运行结果

以上代码的其他情况读者可以通过更改相应变量来查看，这里不再做演示。下面来演示一个嵌套出现歧义的示例。

【示例 3-5】以下代码演示一个会出现歧义的 if 语句嵌套。

```
<?php
    $x=6;                                   //定义并初始化一个变量
    if($x>0)                                //判断变量是否大于 0
        if($x<10)
            echo "0<$x<10";        匹配
    else                                    //当变量不大于 0 时输出
```

```
        echo "$x<=0";
?>
```

代码运行结果如图 3.10 所示。

当将代码中的$x 改为-6 时，运行的结果如图 3.11 所示。

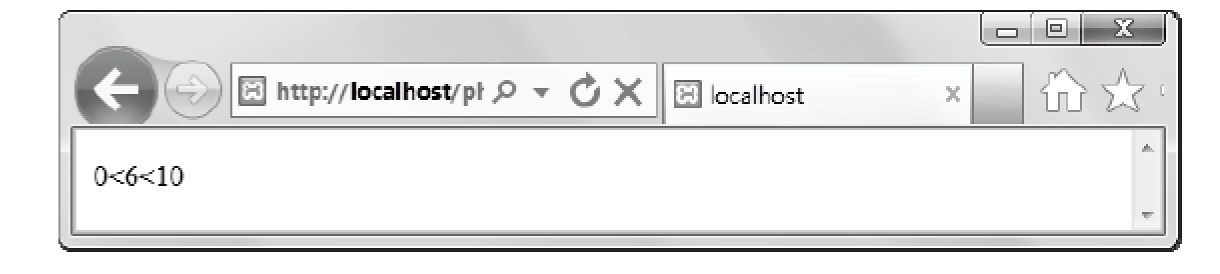


图 3.10　运行结果

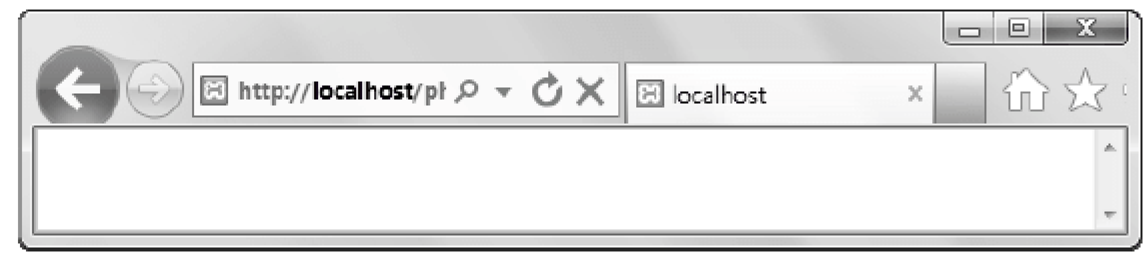


图 3.11　运行结果

可以从上面的运行结果看出，程序并没有输出我们期望的结果。这就是一个明显的嵌套错误，解决的办法就是，将 if 条件执行的语句改为复合语句的形式，修改后的代码如下：

```
<?php
    $x=-6;                                       //定义并初始化一个变量
    if($x>0){                                    //将 if 判断后的代码改为语句块形式
        if($x<10)
            echo "0<$x<10";              匹配
    }
    else                                         //当变量不大于 0 时输出
        echo "$x<=0";
?>
```

代码运行结果如图 3.12 所示。

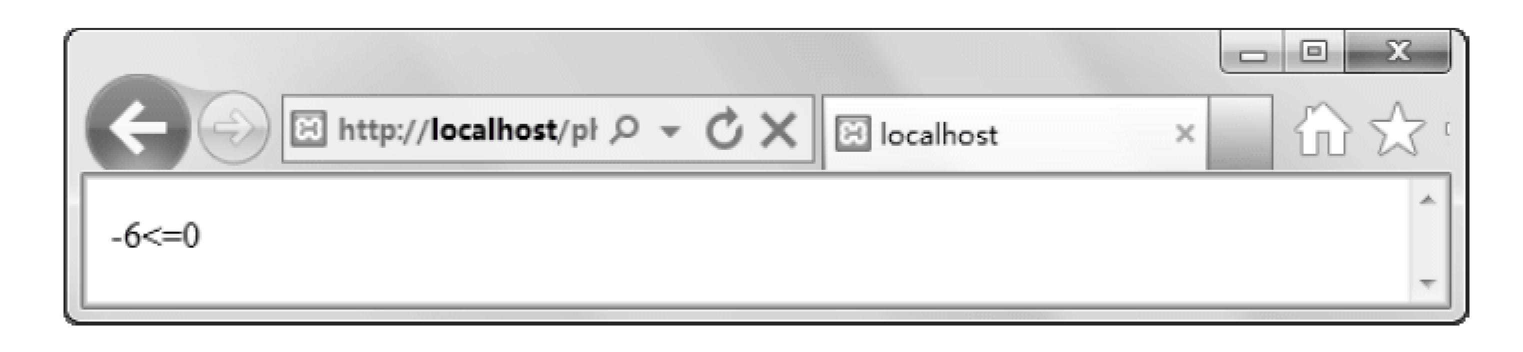


图 3.12　运行结果

我们可以看到，程序运行后就输出了正确的结果。

3.2.2　switch 语句

switch 语句用来实现按照不同的情况执行不同的语句。switch 语句常用于对变量的不同取值执行不同的语句。if…else…if 形式也可以用来实现类似的功能，但是它常用于针对变量的一个范围执行不同的语句，它的一般形式如下：

```
switch (表达式)
{
    case 常量 1:
        语句 1 或空;
        break;
    case 常量 2:
        语句 2 或空;
        break;
    …
    case 常量 n:
        语句 n 或空;
```

```
        break;
    default:
        语句 n+1 或空;
}
```

switch 语句执行时，首先计算表达式的值，并将它与每一个 case 后的常量进行比较。如果与某个常量相等，则执行对应的语句，遇到 break 语句则退出 switch 结构，否则就一直向下执行，直到遇到 break 或者 default。结构中的 default 不是必须有的，它用来匹配 case 情况之外的所有情况。switch 语句的程序流程如图 3.13 所示。

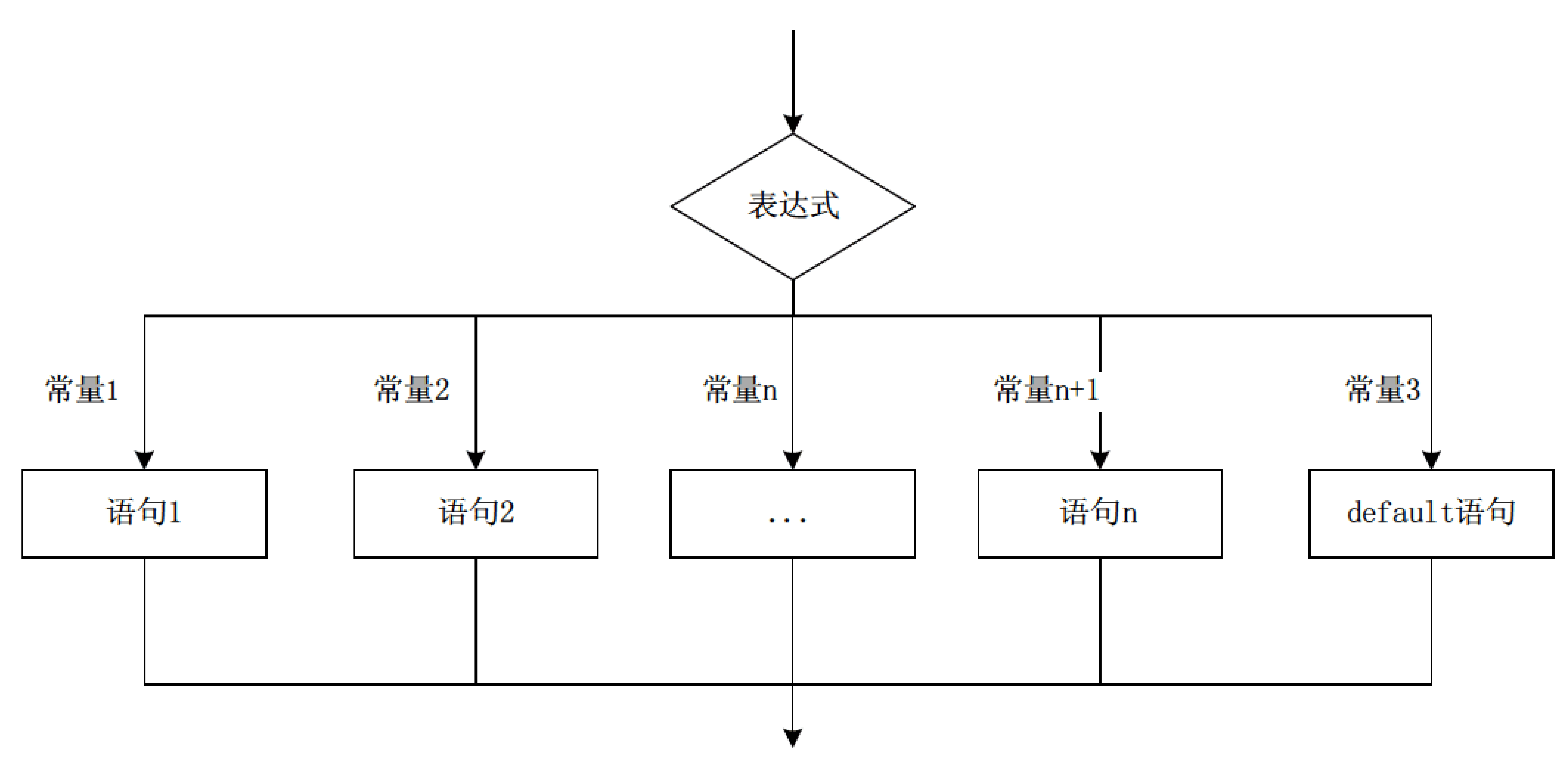


图 3.13　switch 流程图

【示例 3-6】以下代码演示 switch 语句的多种使用形式。

```
<?php
    $week=0;                //定义并初始化星期变量
    switch($week){
        case 0:             //变量为 0 的情况
            echo '星期日。';
            break;
        case 1:             //变量为 1 的情况
            echo '星期一。';
        case 2:             //变量为 2 的情况
            echo '星期二。';
            break;
        case 3:             //变量为 3 的情况
        case 4:             //变量为 4 的情况
            echo '星期三或者星期四。';
            break;
        case 5;             //变量为 5 的情况
            echo '星期五。';
            break;
        default:            //变量为以上 case 之外的所有情况
            echo '星期六。';
    }
?>
```

代码运行结果如图 3.14 所示。

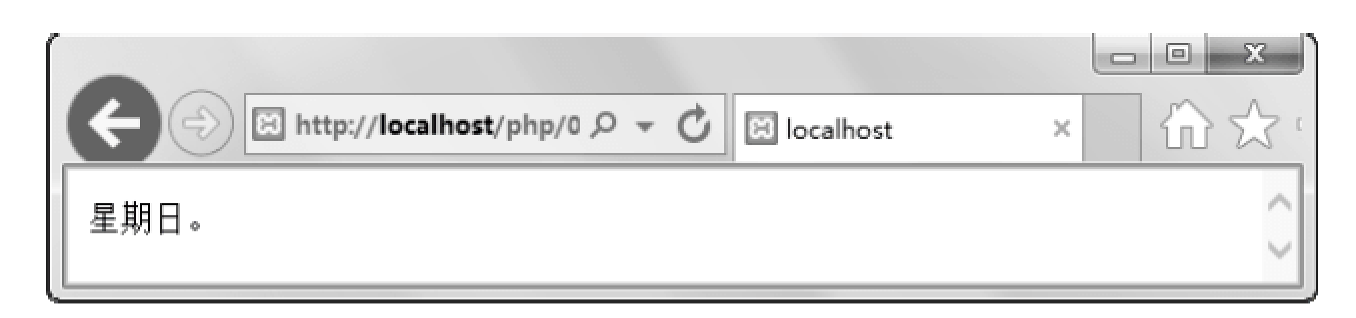


图 3.14　运行结果

以上代码中变量值为 0，因此运行结果会输出变量为 0 的情况。我们将代码中的 week 变量值设置为 1 后的运行结果如图 3.15 所示。



图 3.15　运行结果

以上运行结果输出了两个星期，是因为代码中 case 下的语句中没有 break 语句，因此程序无条件执行变量为 2 的情况下语句并未遇到 break 后停止。我们再将代码中 week 变量设置为 3 或者 4 后，运行结果如图 3.16 所示。

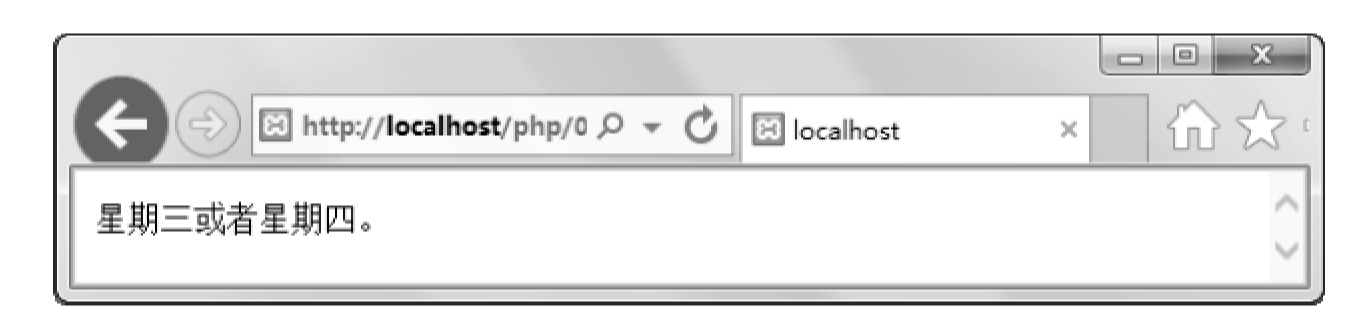


图 3.16　运行结果

变量为 3 或者 4 的运行情况是相同的，因为变量为 3 的情况下没有任何语句，因此会无条件执行后面情况下的语句，直到遇到 break。我们再将变量改为除 0～5 之外的任意类型值，运行程序后会输出如图 3.17 所示的结果，因为这些值都会被 default 匹配。

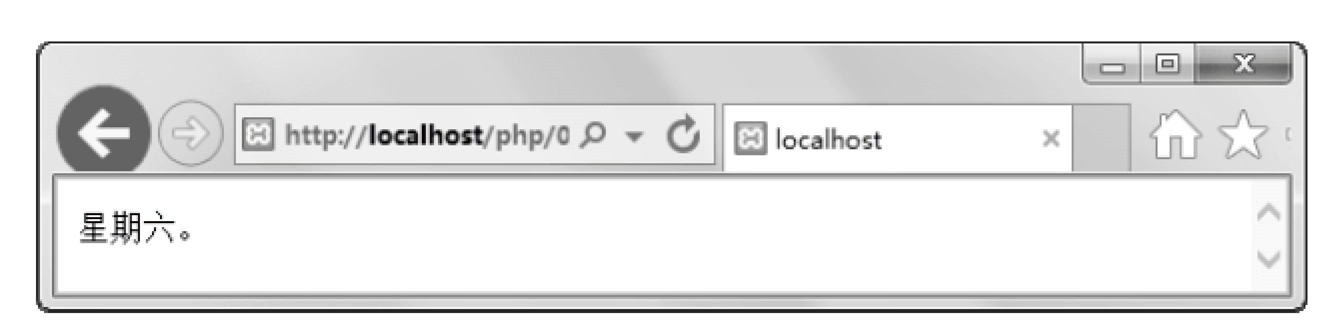


图 3.17　运行结果

在示例 3-6 简短的代码中就包含了 switch 的多种用法和技巧，读者应该多更换变量的值来查看输出结果并记住这些技巧。

3.3　循 环 语 句

计算机最擅长做的工作就是重复地执行系列的命令。循环语句就是指定一系列的语句并规定一个条件，让计算机重复执行这些语句直到满足规定的条件为止。PHP 中提供了 for 循环、while 循环和 do…while 循环，下面就来介绍这些知识。

3.3.1　for 循环

for 循环使用灵活性比较高，是 PHP 中使用最频繁的循环语句。for 循环的一般形式如下：

```
for （表达式 1； 表达式 2； 表达式 3）
{
  语句;
}
```

表达式 1 通常为赋值语句，用来初始化循环控制变量的初始值；表达式 2 通常为关系表达式或者逻辑表达式，用来确定何时停止循环；表达式 3 通常为递增或者递减表达式，用来对循环控制变量进行修改以逐步不满足表达式 2 的条件，否则就有可能造成无限循环。for 循环条件的流程图如图 3.18 所示。

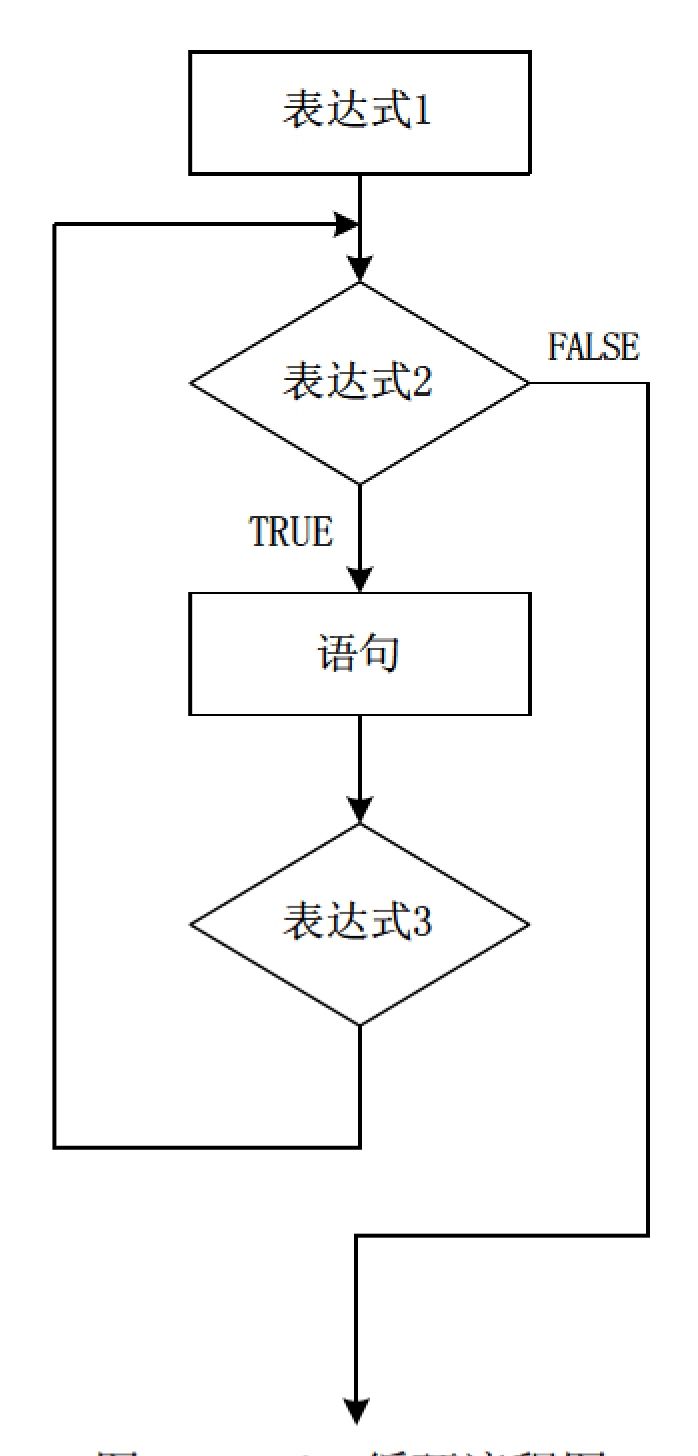


图 3.18　for 循环流程图

for 循环的执行过程如下：

（1）计算表达式 1 的值，为循环控制变量赋初值，该语句只在循环开始时执行一次。

（2）计算表达式 2 的值，如果其值为 TRUE，则执行循环体语句，否则退出循环。

（3）在每一次执行循环体语句结束后，运行一次表达式 3，以调整循环控制变量。然后返回第（2）步重新计算表达式 2 的值，依次重复，直到表达式 2 的条件不成立为止。

【示例 3-7】以下代码演示使用 for 循环输出数字 1～5。

```
<?php
    for($i=1;$i<=5;$i++){
        echo "$i<br />";        //循环体
    }
?>
```

代码运行结果如图 3.19 所示。

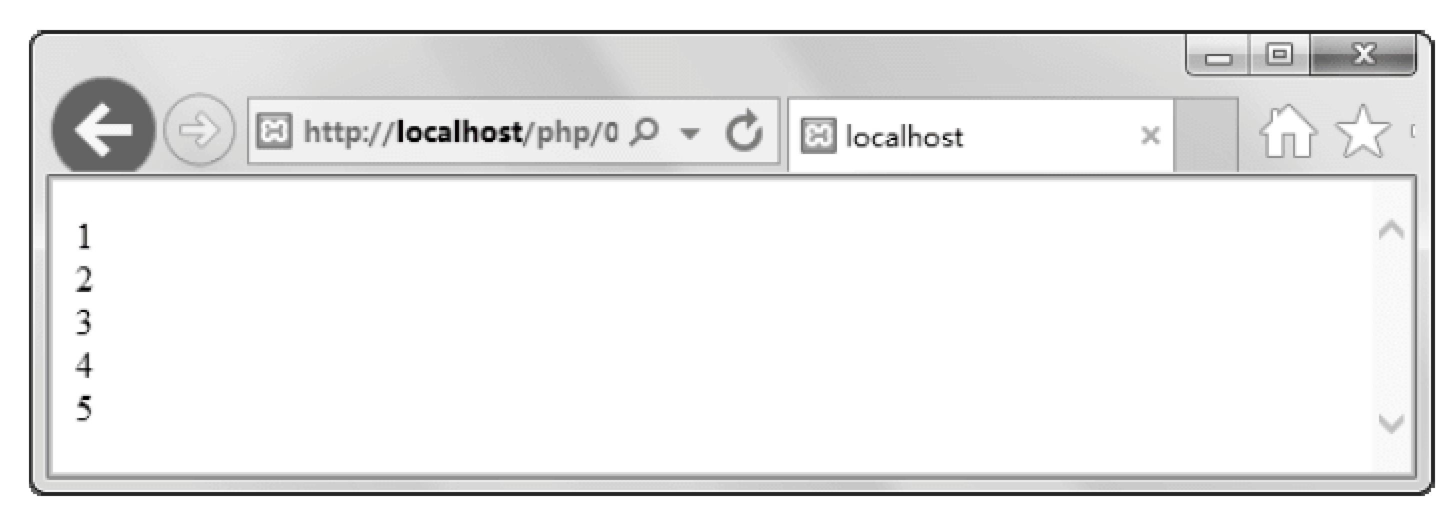


图 3.19　运行结果

for 循环的灵活之处就表现在它可以有多种形式，下面我们来分别介绍这些形式。

- ❑ 省略表达式 1：这种形式的表达式 1 通常写在 for 循环的外面。这种形式主要用在循环控制变量的初值不是已知常量，而是需要通过前面语句的执行计算得到的情况。

【示例 3-8】以下代码演示 for 循环省略表达式 1 的使用。

```
<?php
    $x=10;                      //初始化两个变量
    $y=5;
    $z=$x-$y;                   //初始化变量
    for(;$z<=5;$z++){           //省略表达式 1 的 for 循环
        echo "$z<br />";
    }
?>
```

代码运行结果如图 3.20 所示。

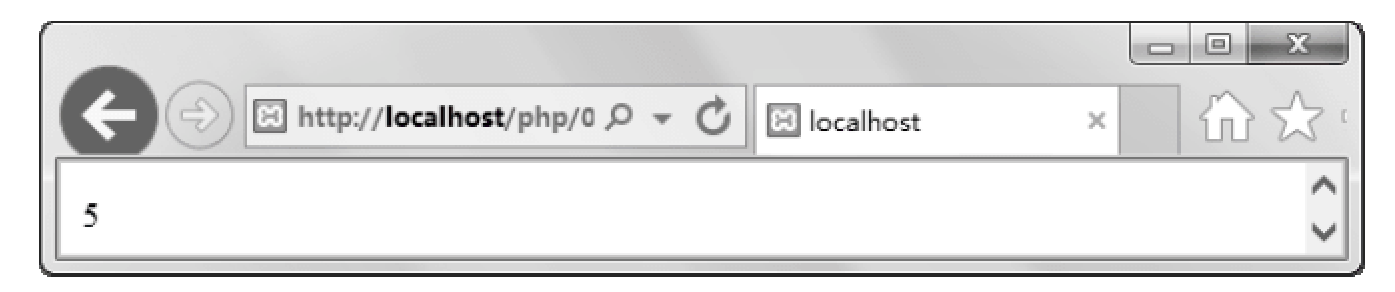


图 3.20　运行结果

将以上代码中的$y 的值改为 7 后，运行结果如图 3.21 所示。

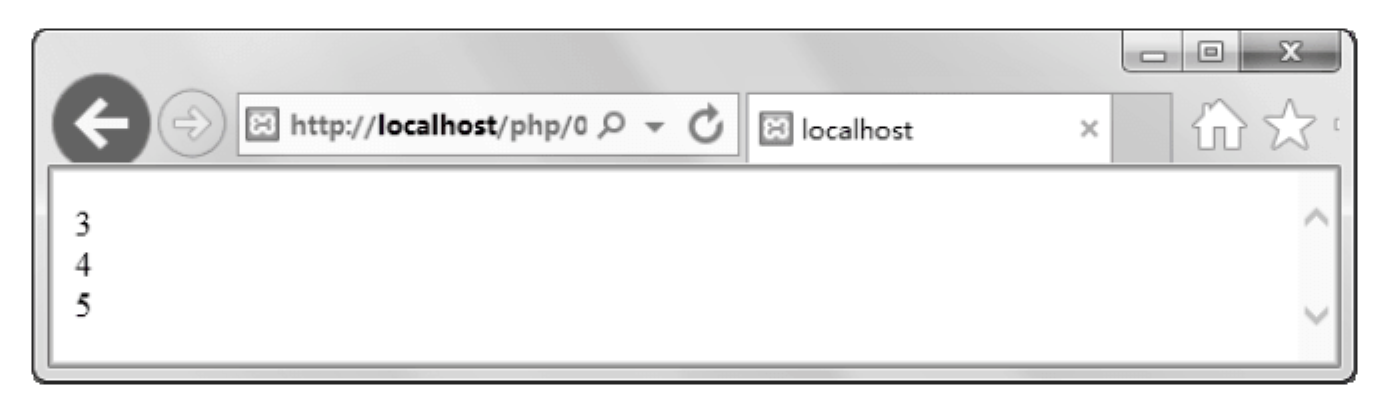


图 3.21　运行结果

以上形式就是通过外部的运算来确定表达式 1 的值进而影响到 for 循环的次数。

- 省略表达式 2：因为表达式 2 默认值为 TRUE，因此这种形式如果在循环体中不加入跳转语句将会是一个无限循环。
- 省略表达式 3：这种形式的表达式 3 通常写在循环体内，在循环体内改变表达式的值，常用的形式如下所示。

```
<?php
    for($i=0;$i<=100;){         //省略表达式 3
        …
        $i=$x*3+4;              //在内部修改其值
        …
    }
?>
```

上面介绍的是省略 for 循环中表达式的情况。for 循环中不仅每个表达式可以为空，并且每个表达式还可以是由多个逗号分隔的表达式构成。

【示例 3-9】以下代码演示每个表达式为多个逗号分隔的表达式的情况。

```
<?php
    for($x=1,$y=2,$z=3;$x<=1,$y<=3,$z<=5;$x++,$y++,$z++){
                                        //多个逗号分隔的表达式
        echo "$x$y$z<br />";            //输出三个变量的值
    }
?>
```

代码运行结果如图 3.22 所示。



图 3.22　运行结果

从图 3.22 的运行结果可知，程序循环只进行了 3 次，这是因为虽然 for 循环中的表达式 2 所有用逗号分隔的表达式都会计算，但只取最后一个结果，因此$z 从 3 到 5 只用 3 次循环即可。

3.3.2　while 循环

while 循环与 for 循环相比使用比较简单，它通常用于循环次数不确定的情况，其一般形式如下：

```
while (表达式)
{
  语句;
}
```

while 循环在开始和每次执行循环体语句后均会判断表达式的值，如果为 TRUE，则执行循环体，如果为 FALSE 则退出 while 循环，它的执行流程图如图 3.23 所示。

【示例 3-10】以下代码演示使用 while 循环输出数字 1～5。

```
<?php
    $x=1;                    //初始化变量
    while($x<=5){            //执行 while 循环
        echo "$x<br />";
        $x++;
    }
?>
```

代码运行结果如图 3.24 所示。

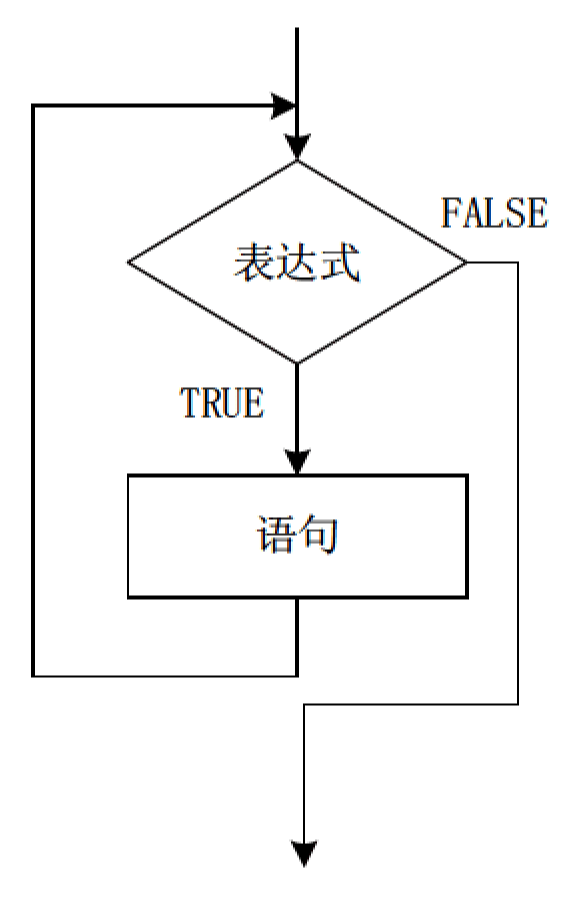


图 3.23　while 循环流程图

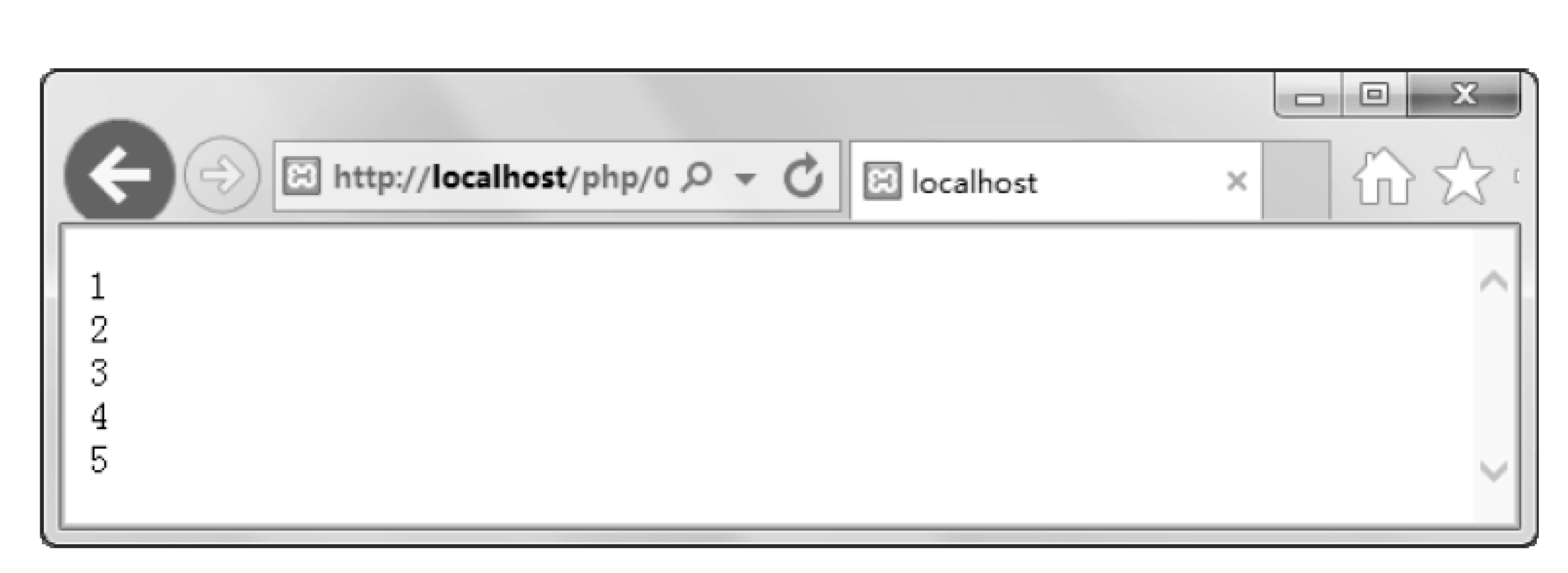


图 3.24　运行结果

我们使用 while 循环同样实现了输出数值 1～5。这里读者需要明白的一点是，通常循环语句间是可以进行转换的。

3.3.3　do…while 循环

do…while 循环与 while 类似，它的一般形式如下：

```
do
{
  语句;
}
while (表达式)
```

do…while 循环会首先执行一次循环体中的语句，然后再去判断表达式中的条件，如果为 TRUE 则继续执行循环体，如为 FALSE 则退出循环。也就是说，do…while 循环会保证循环体被执行一次，它执行的流程图如图 3.25 所示。

【示例 3-11】以下代码演示 do…while 循环的使用。

```
<?php
    $x=10;          //初始化变量 x
    do{
        echo "$x";
    }while($x<5);
?>
```

代码运行结果如图 3.26 所示。

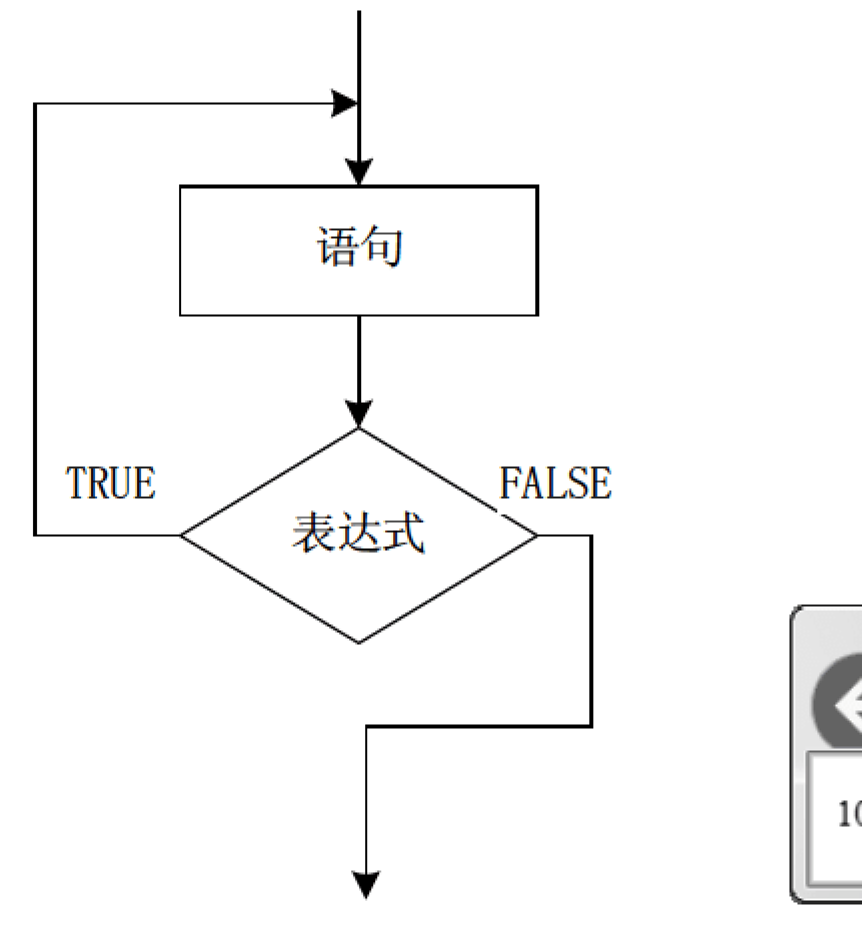


图 3.25　do…while 循环流程图

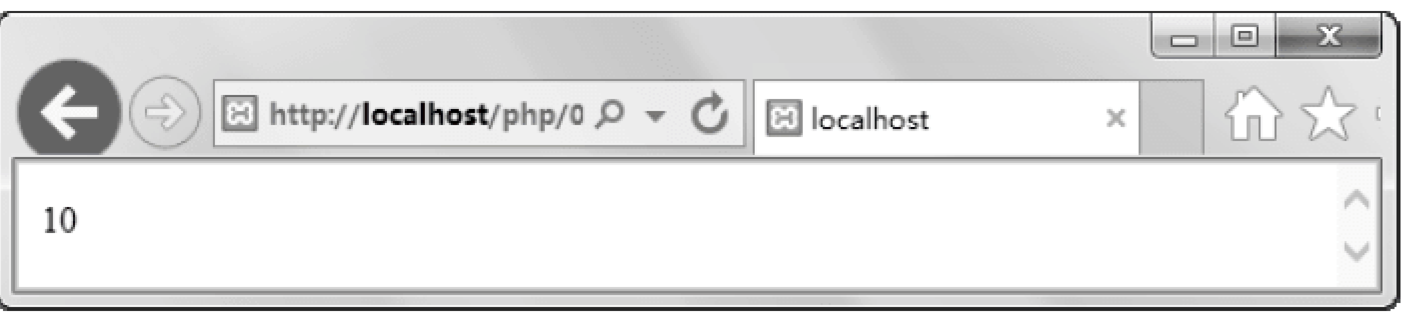


图 3.26　运行结果

在代码中的判断条件虽然为 FALSE，但是循环体语句仍会执行一次，这就是 do…while 循环的特性。

除了以上介绍的 3 种循环语句之外，PHP 还支持 foreach 循环语句，它是专门用来操作数组的循环语句，因此该循环将在数组学习部分进行讲解。

3.3.4　循环语句的嵌套

同选择语句类似，循环语句也是可以嵌套的，这里我们以实现如图 3.27 所示的输出来

编写代码。

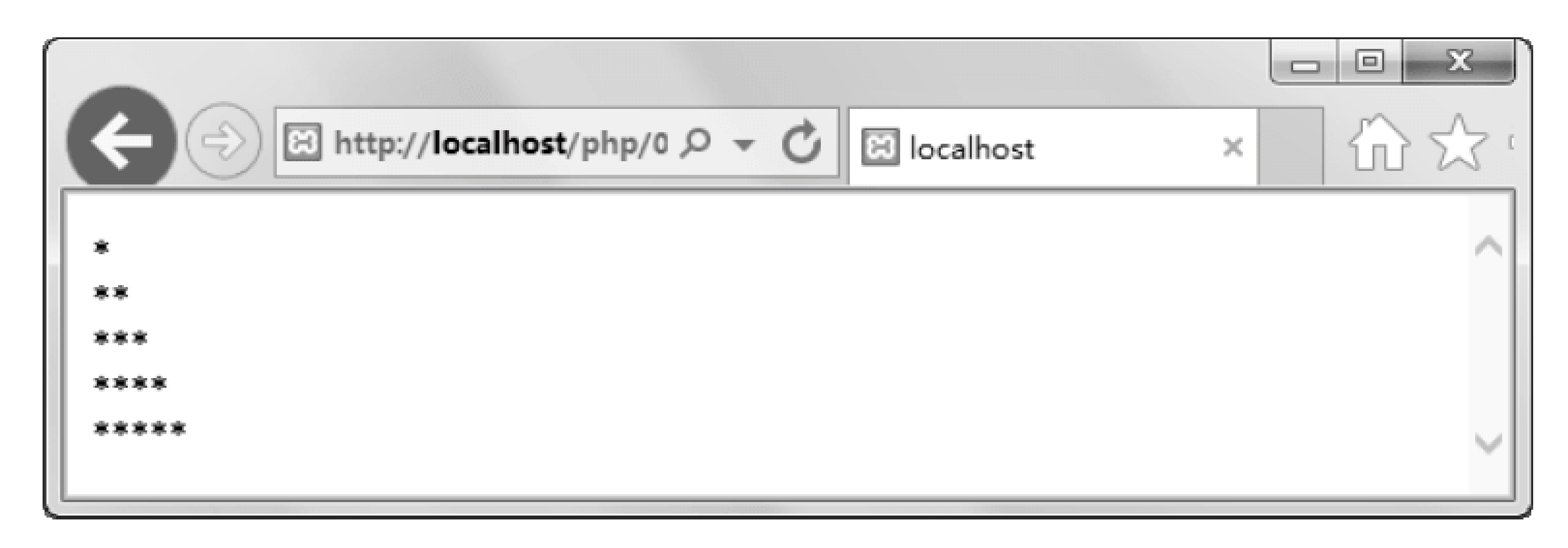


图 3.27　运行结果

上面输出的实现还是比较简单的，它的规律就在于在对应的行输出对应行数的星号（*），然后在输出完毕后加入一个换行符即可。因此就可以通过 for 循环来每循环一次输出一个换行符，代码如下：

```
for(;;){
    echo '<br />';
}
```

每次循环开始，我们就可以在换行前输出对应的星号，这里同样使用 for 循环来实现，代码如下：

```
for(;;){
    echo '*';
}
```

由于输出输出的行数和星号数相同，因此可以通过这个关系将两个循环联系起来，简略代码如下：

```
for($i=$line;;){
    for(;$j=$i;){
        echo '*';
    }
    echo '<br />';
}
```

其中的变量 line 用来表示输出的行数，我们可以定义在循环之外用来修改行数。完整的代码如下：

```
<?php
    $line=5;                        //用来控制行数
    for($i=1;$i<=$line;$i++){
        for($j=1;$j<=$i;$j++){
            echo '*';               //输出星号
        }
        echo '<br />';              //输出换行
    }
?>
```

运行以上代码即可输出本节开始的效果图，读者可以通过改变 line 变量来控制输出的行数。

3.4　跳 转 语 句

跳转语句用于跳出一个循环结构或者跳转到指定的位置执行。PHP 中常用的跳转语句包括 break、continue、return 和 goto，下面就分别介绍这些语句。

3.4.1　break 语句

break 语句我们在学习 switch 语句时就多次使用过，它用来当前 for、foreach、while、do…while 或者 switch 结构的执行。break 可以接受一个可选的参数用来指定 break 跳出几层语言结构，默认值为 1，常用在循环嵌套中。

【示例 3-12】以下代码演示使用 break 语句跳出无限循环。

```
<?php
    for($x=1;;$x++){                  //省略表达式 2 的 for 循环将是无限循环
        echo "$x<br />";
        if($x==5){
            break;                    //使用 if 语句控制退出无限循环
        }
    }
?>
```

代码运行结果如图 3.28 所示。

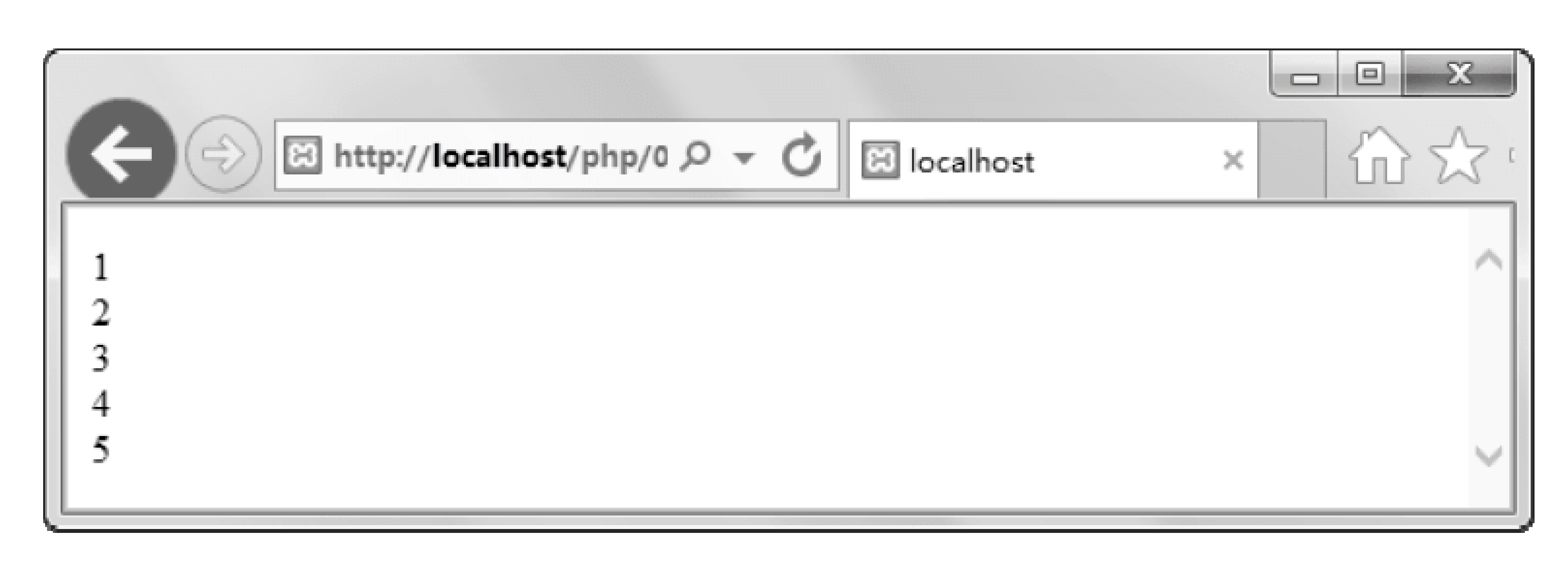


图 3.28　运行结果

从以上运行结果可以看出，这个无限循环并没有使浏览器崩溃，而是正确输出了结果，这也是 break 语句的常用方式。

【示例 3-13】以下代码演示使用 break 语句跳出多层结构。

```
<?php
    for($x=1;$x<=10;$x++){              //第一层 for 循环结构
        echo '1';
        for(;;){                        //第二层 for 循环结构
            echo '<br />2';
            break 2;                    //跳出两层循环结构
        }
    }
?>
```

代码运行结果如图 3.29 所示。

以上代码通过 break 跳出了两个简单的 for 循环结构，在实际应用中，读者应该因地制宜灵活运用 break。



图 3.29　运行结果

3.4.2　continue 语句

continue 语句用来跳过本次循环中剩余的代码，直接进行下一次循环表达式的判断。与 break 语句类似，continue 语句也可以接受一个可选的参数来决定跳过多层结构。

【示例 3-14】以下代码演示使用 continue 语句跳过 3 的倍数数值的输出。

```
<?php
    for ($i=1;$i<=10;$i++) {                    //for 循环输出数值
        if($i%3==0)                             //判断变量是否为 3 的整数倍
            continue;                           //跳过本次循环剩余语句
        echo "$i<br />";                        //输出变量的值
    }
?>
```

代码运行结果如图 3.30 所示。

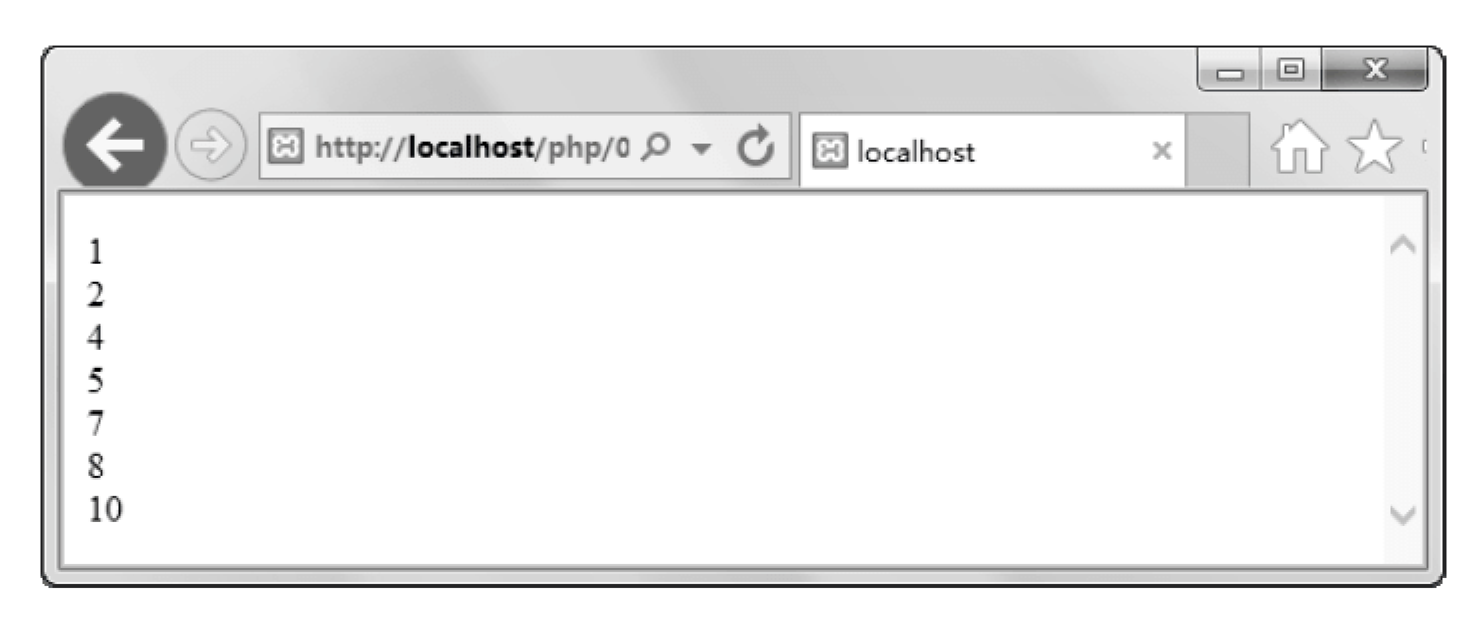


图 3.30　运行结果

以上代码通过循环输出 1～10 直接的数值，由于中间加入了跳转语句，致使是 3 的整数倍的数值不被输出。

【示例 3-15】以下代码演示使用 continue 语句跳出多层结构。

```
<?php
    $i=0;                                       //初始化循环控制变量
    while($i++<3){
        echo '第一层 while 循环。<br />';          //第一层循环输出
        while(TRUE){
            echo "第二层 while 循环。<br />";       //第二层循环输出
            while(TRUE){
                echo "第三层 while 循环。<br />";   //第三层循环输出
                continue 3;
            }
            echo "这里不会被输出<br />";   //由于 continue 控制语句跳出多层
                                            循环，因此这里不会被输出
        }
        echo "这里同样不会被输出。";         //由于 continue 控制语句跳出多层
                                            循环，因此这里不会被输出
    }
```

```
?>
```

代码运行结果如图 3.31 所示。

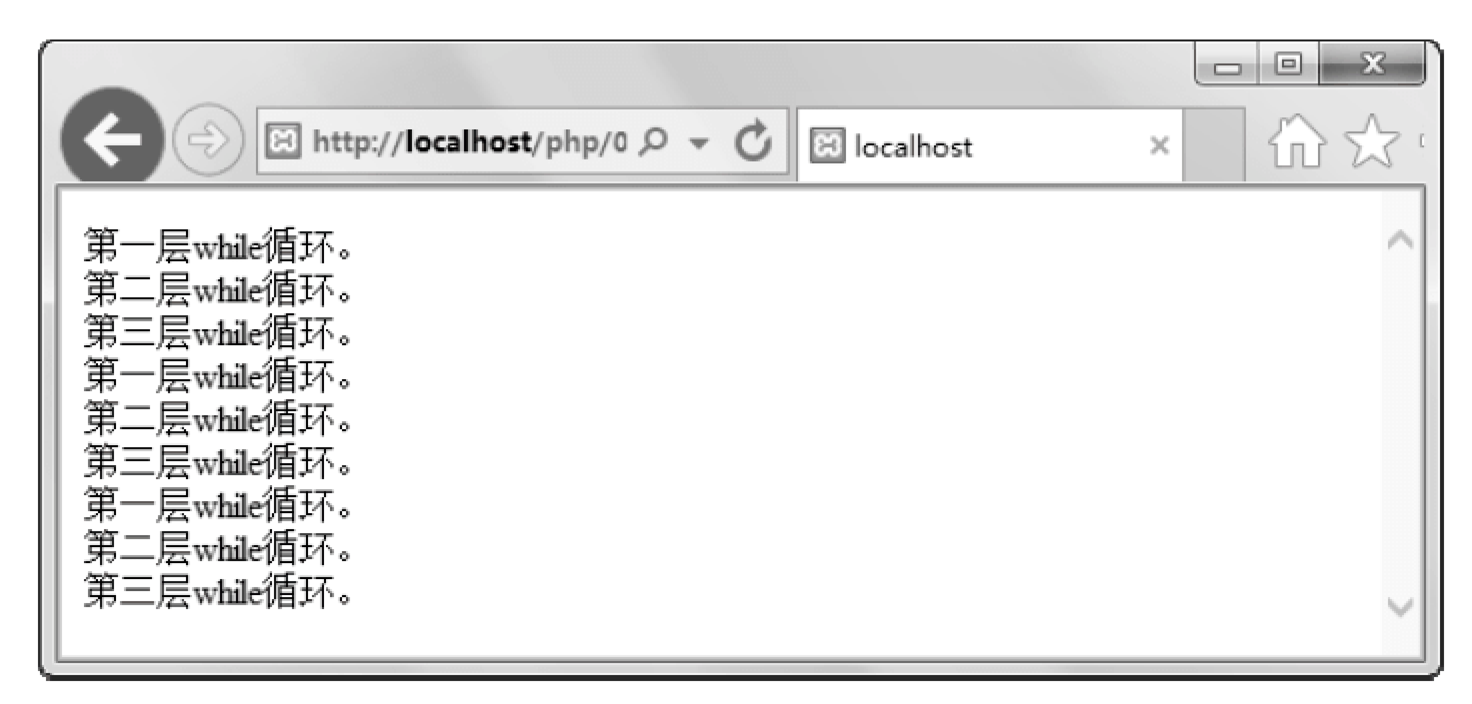


图 3.31　运行结果

3.4.3　goto 语句

goto 语句用来跳转到程序的指定位置开始执行，可以替代 break 跳出多层结构。PHP 中的 goto 语句被限制在只能在同一个文件和作用域中跳转。例如不可以跳入任何循环和 swith 结构中。goto 语句的常用形式如下：

```
goto tab
…
tab:
…
```

tab 用来表示程序要跳转到位置的标记与 goto 后的标记要一致。

【示例 3-16】以下代码演示使用 goto 语句使程序跳转执行。

```
<?php
    for($i=1;$i<=5;$i++){              //使用 for 循环循环输出 1~5
        if($i==3)
            goto ECH;                  //当$i 为 3 时候跳出 for 循环
        echo "$i<br />";
    }
    ECH:                               //执行 goto 语句后程序将从此处开始执行
    echo "此时\$i=$i";
?>
```

代码运行结果如图 3.32 所示。



图 3.32　运行结果

我们可以看到 goto 语句的使用非常方便，但是在编程过程中应该尽量少使用，过多地使用 goto 语句会使程序流程变得混乱，进而使程序阅读和调试都变得非常困难。PHP 中除了上述 3 种跳转语句之外，return 语句也是用来使程序进行跳转，return 语句的知识将在函

数的学习部分进行讲解。

3.5　小　　结

本章主要学习了构成程序整体框架的语言结构，这些知识包括语句的基本概念、选择语句、循环语句和跳转语句。其中的循环语句部分是本章的重点，而且由于循环语句的执行流程均有各自的特点，掌握起来也是比较难的。因此读者应该多实践，从实践中理解这些知识。

3.6　本 章 习 题

1. 使用 if…else 判断一个数是否大于等于 0 并输出如图 3.33 所示的结论。



图 3.33　运行效果

2. 使用 while 循环输出数字 1～9，执行结果应该类似图 3.34 所示。
3. 使用 for 循环输出 1～20 之间的偶数，执行结果应该类似图 3.35 所示。

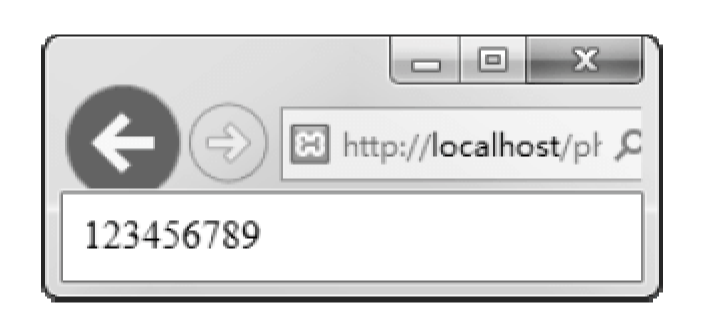


图 3.34　运行效果



图 3.35　运行效果

4. 使用任意一种循环，输出如图 3.36 所示的九九乘法表。

http://localhost/php/xiti/03/3.4.php　localhost

1×1=1								
1×2=2	2×2=4							
1×3=3	2×3=6	3×3=9						
1×4=4	2×4=8	3×4=12	4×4=16					
1×5=5	2×5=10	3×5=15	4×5=20	5×5=25				
1×6=6	2×6=12	3×6=18	4×6=24	5×6=30	6×6=36			
1×7=7	2×7=14	3×7=21	4×7=28	5×7=35	6×7=42	7×7=49		
1×8=8	2×8=16	3×8=24	4×8=32	5×8=40	6×8=48	7×8=56	8×8=64	
1×9=9	2×9=18	3×9=27	4×9=36	5×9=45	6×9=54	7×9=63	8×9=72	9×9=81

图 3.36　九九乘法表

第 4 章　函　　数

函数是由一系列语句组成的一个功能独立且完整的代码块。它可以用来完成一些指定的操作，并返回操作后的结果。使用函数可以简化代码结构，并降低代码编写工作量。本章将主要介绍使用函数的优势和函数的使用方法。

4.1　使用函数的优势

使用函数有两个最主要的优势：函数是一个完整而且独立的功能块，因而使用函数的代码结构更加清晰，可以明显增强代码的可读性；函数使得程序的可复用性非常强。PHP 系统默认已经定义了多种类型的千余个函数，这些函数可以被任何一个 PHP 程序使用，这就是函数可复用性的最明显表现。

4.2　使 用 函 数

PHP 中的函数分为两类，分别是系统函数和用户自定义函数。系统函数的使用比较简单，本节将重点介绍自定义函数的知识。

4.2.1　自定义函数和调用函数

自定义函数即为用户自己定义用来实现指定需求的函数。定义函数的一般形式如下：

```
function 函数名(参数列表){
    函数体;
    return 返回值;
}
```

定义函数需要使用 function 关键字；函数名是用来为这一系列操作定义的名称；参数列表是数据传入函数的入口；函数体是函数的注意功能实现部分；函数都有返回值，但在没有显式调用 return 语句返回一个值的时候，函数会默认返回空类型（void）。

【示例 4-1】下面就来定义一个函数用来实现两个数相加并返回操作结果的函数。其所示如下：

```
function add($x,$y){
    return $x+$y;            //返回变量之和
}
```

以上代码中，add 是函数名；$x 和$y 是函数接受的参数；return 后的表达式即为函数的返回值。函数调用的一般形式如下：

```
函数名(参数列表);
```

函数名即在函数定义时指定的名称，参数列表需要对应函数定义时的参数列表传入。

【示例 4-2】以下代码演示使用自定义函数 add()计算两数之和。

```
<?php
    function add($x,$y){                          //定义函数 add()
        return $x+$y;
    }
    echo add(5,6);                                //调用函数并输出结果
?>
```

代码运行结果如图 4.1 所示。

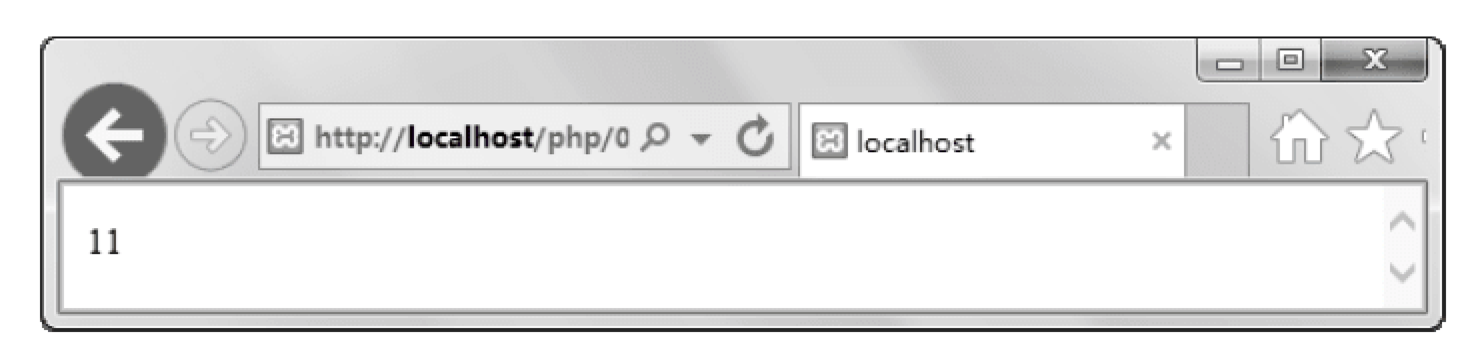


图 4.1　运行结果

我们可以看到只要通过使用正确定义的函数名，就可以轻易得到相应的结果。

在上面的学习中，我们已经认识到 return 可以将函数的运算结果返回。return 语句也是跳转语句的一种，它会立刻终止当前函数的执行并返回结果。因此，return 语句通常会放在整个函数体的最后。

4.2.2　函数的参数

函数的参数有实际参数和形式参数之分。形式参数即为一个形式，它只要可以保证参数能正确地传递到函数体中即可。因此，只要符合 PHP 命名规范的名称都可以使用。以下函数的功能是完全一致的。

```
function add($x,$y){
    return $x+$y;
}
function add($a,$b){                //使用不同的形式参数
    return $a+$b;
}

function add($abc,$def){            //使用不同的形式参数
    return $abc+$def;
}
```

函数在定义时可以为形式参数设定默认参数，默认参数的定义必须是从右向左，并且默认参数的右边不可有为设置默认参数的形式参数。以下是一些默认参数的示例。

```
function func($x,$y,$z=0){              //正确的默认参数设置方法
    …
}

function func($x,$y=1,$z=0){            //正确的默认参数设置方法
    …
}
```

```
function func($x=2,$y=1,$z=0){            //正确的默认参数设置方法
    …
}
function func($x=2,$y=1,$x){              //错误的默认参数设置方法
    …
}
function func($x=0,$y=1,$z){              //错误的默认参数设置方法
    …
}
```

【示例 4-3】以下代码演示为 add()函数设置默认参数后的运行效果。

```
<?php
    function add($x=2,$y=3){                  //定义函数并设置默认参数
        return $x+$y;
    }
    echo add();                               //不传入参数调用 add()函数
    echo '<br />'.add(5);                     //传入一个参数调用 add()函数
    echo '<br />'.add(5,6);                   //传入两个参数调用 add()函数
?>
```

调用　$x=5　$x=5 $y=6

代码运行结果如图 4.2 所示。

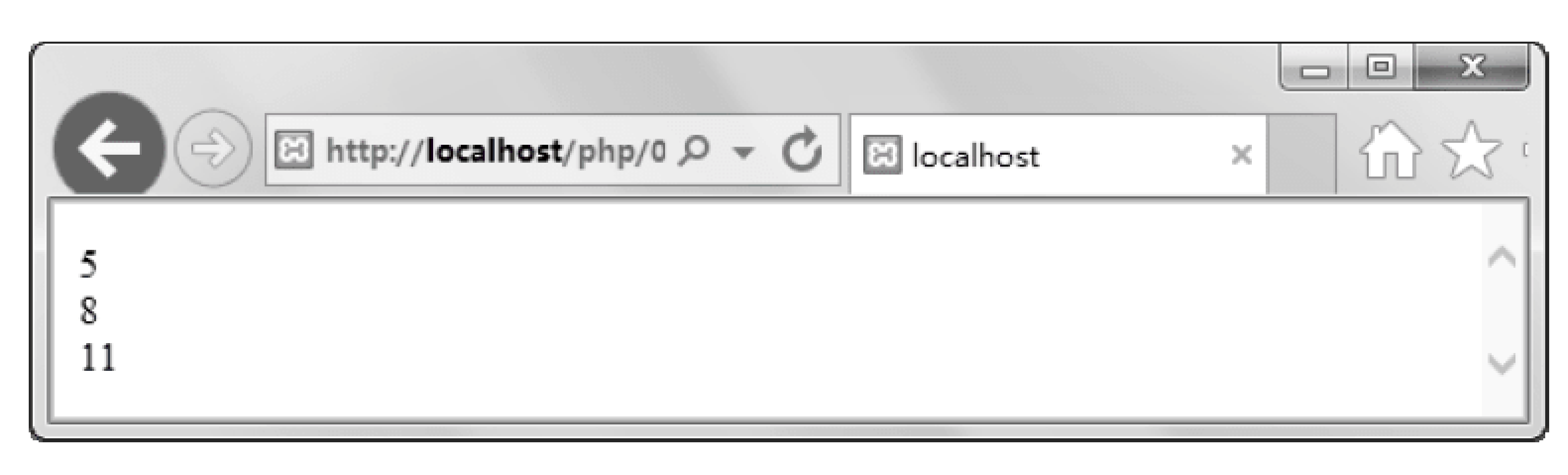


图 4.2　运行结果

在调用的时候，传入的参数即为实际参数，它会从左向右依次传入函数的参数列表，设置默认值的参数将被替换。

4.2.3　参数的传递

参数的传递类似于变量的赋值。PHP 参数传递方式分为按值传递和按引用传递。类似地，按值传递将传递实际参数的副本，而引用传递则传递实际参数本身。

【示例 4-4】以下代码演示使用按值传递方式交换两个数值。

```
<?php
    function swap($x,$y){                         //定义交换数值函数
        $temp=$x;
        $x=$y;
        $y=$temp;
    }
    $m=5;
    $n=15;
    echo "交换前：<br />\$m=$m<br />\$n=$n";      //输出交换前变量的值
    swap($m,$n);                                  //调用函数交换变量数值
    echo "<br />交换后：<br />\$m=$m<br />\$n=$n";   //输出交换后变量的值
?>
```

$x=$m　$y=$n

代码运行结果如图 4.3 所示。

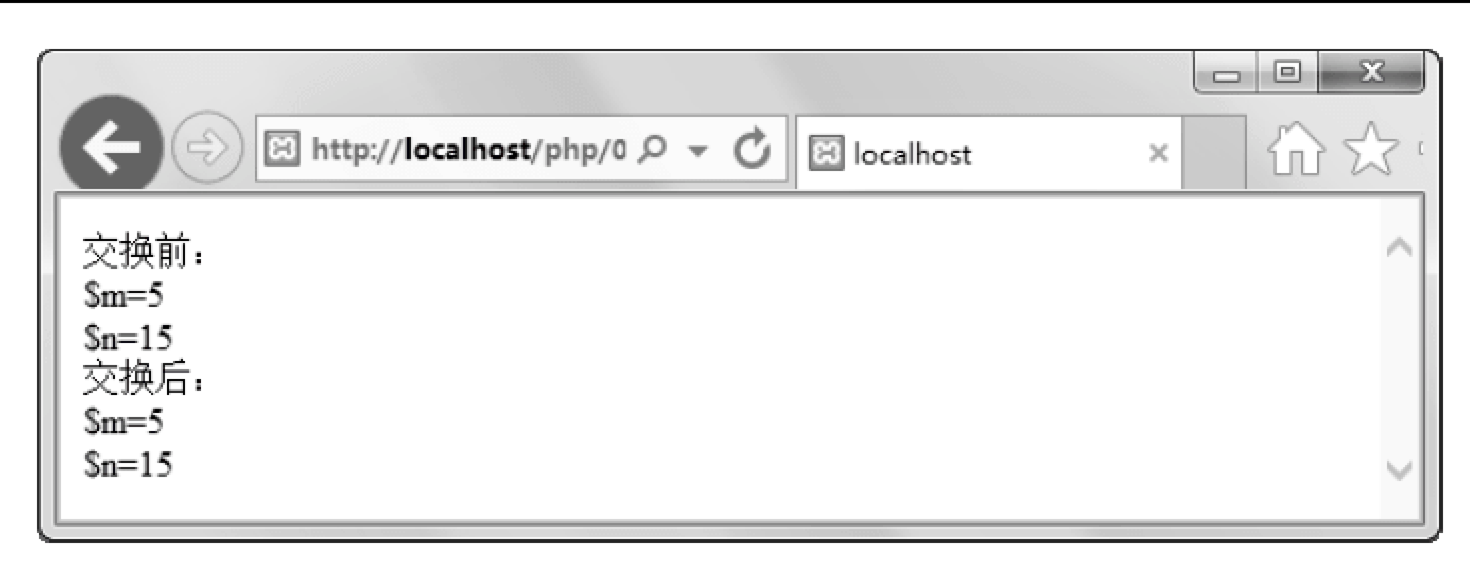


图 4.3　运行结果

从运行结果可以很明显地看出，变量的值并没有被交换。而在函数内部确实实现了变量值的交换，我们可以通过以下代码来验证：

```
<?php
    function swap($x,$y){                                      //定义交换数值函数
        echo "交换前：<br />\$x=$x<br />\$y=$y<br />";  //输出交换前的变量值
        $temp=$x;                                       $x=5
        $x=$y;                                          $y=15
        $y=$temp;
        echo "交换后：<br />\$x=$x<br />\$y=$y";         //输出交换后的变量值
    }
    swap(5,15);                                          //调用函数并传入参数
?>
```

代码运行结果如图 4.4 所示。

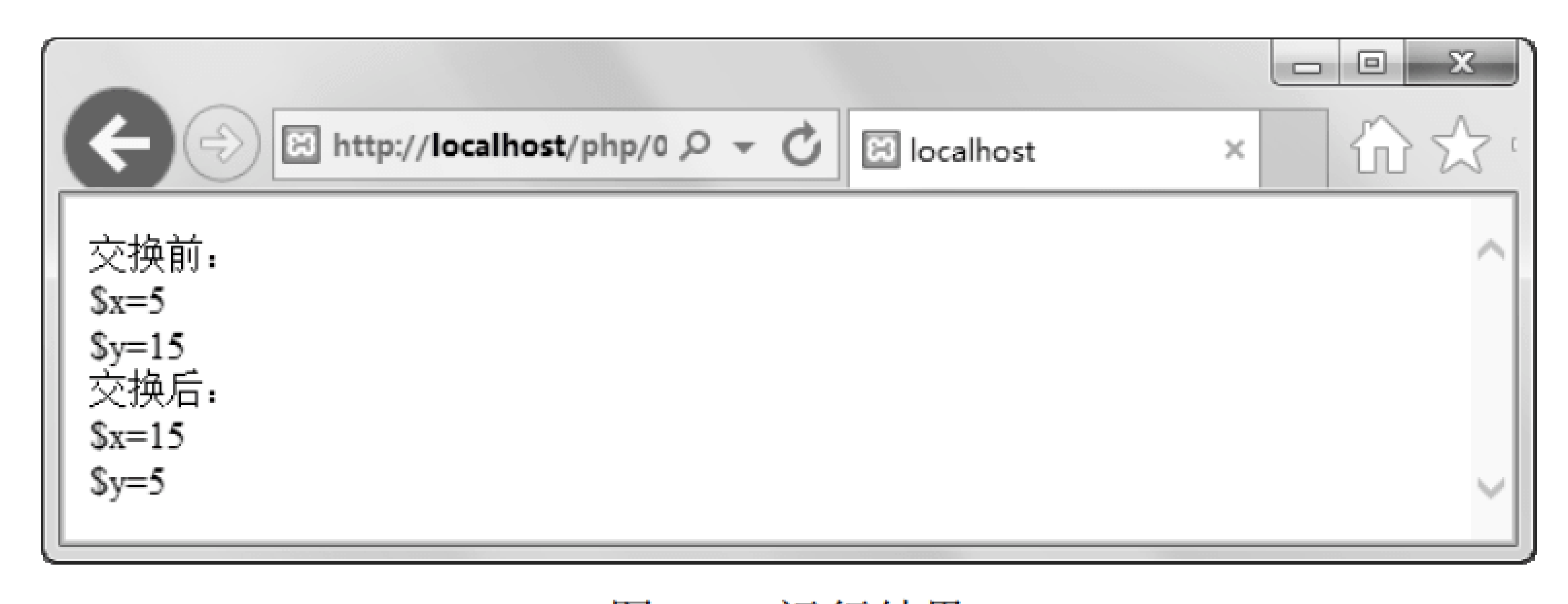


图 4.4　运行结果

从以上输出函数内部交换过程的运行结果来看，在函数内部确实成功交换了变量的值。这就再次证实了按值传递方式中函数操作的是实际参数的副本。类似于变量的赋值，我们同样可以将变量的本身作为参数传递给函数体，这类函数定义的方式如下：

```
function 函数名(&参数1,&参数2,&参数…){
    函数体;
    return 返回值;
}
```

【示例 4-5】以下代码演示将变量本身传递给交换变量值的函数。

```
<?php
    function swap(&$x,&$y){                    //定义交换数值函数
        $temp=$x;
        $x=$y;
        $y=$temp;
    }
    $m=5;
```

```
    $n=15;
    echo "交换前：<br />\$m=$m<br />\$n=$n";        //输出交换前变量的值
    swap($m,$n);                                     //调用函数交换变量数值
    echo "<br />交换后：<br />\$m=$m<br />\$n=$n";//输出交换后变量的值
?>
```

代码运行结果如图 4.5 所示。

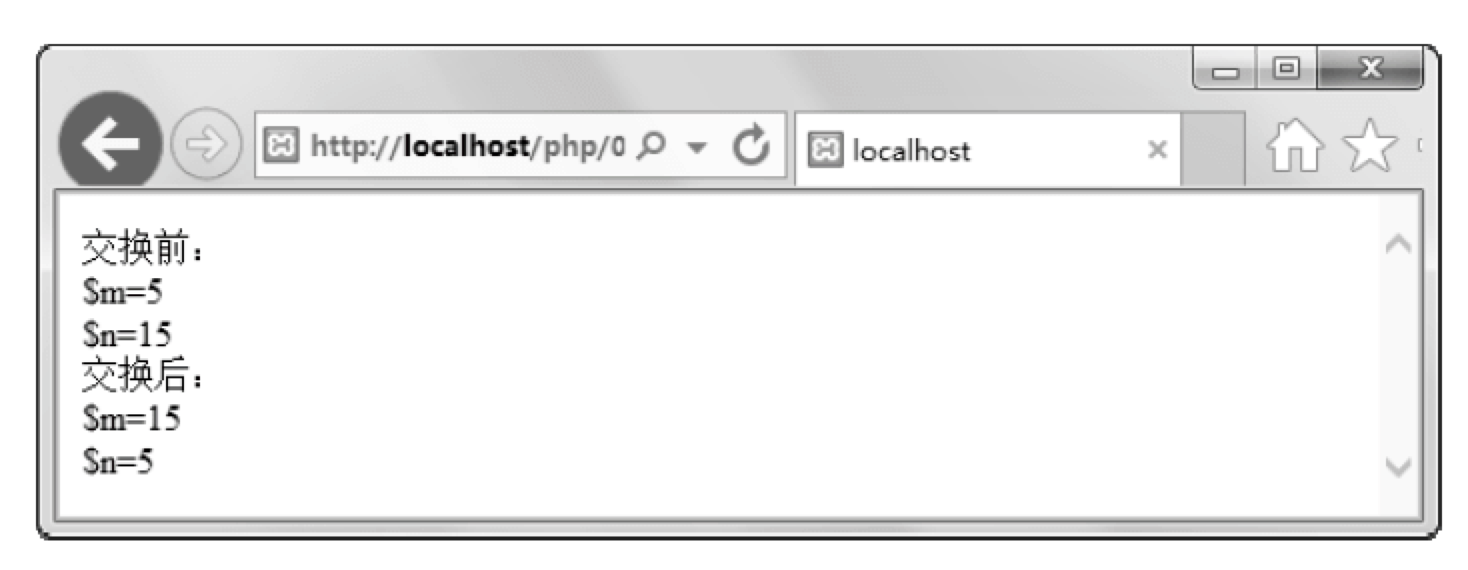


图 4.5　运行结果

从运行结果可以看出，函数成功交换了变量的值。

4.2.4　变量的作用域

变量的作用域即变量可使用的范围。函数中变量的作用是即整个函数体内，因此在函数内部与外部的同名变量是不同的。

【示例 4-6】以下代码演示在函数内和函数外定义同名变量并输出它们的和。

```
<?php
    function add(){           //定义函数 add()
        $m=2;                 //函数内部定义并初始化两个变量
        $n=3;
        return $m+$n;
    }
    $m=12;                    //函数外部定义两个同名变量
    $n=8;
    echo '调用函数输出结果：'.add();
    echo '<br />函数外部输出结果：'.($m+$n);
?>
```

代码运行结果如图 4.6 所示。

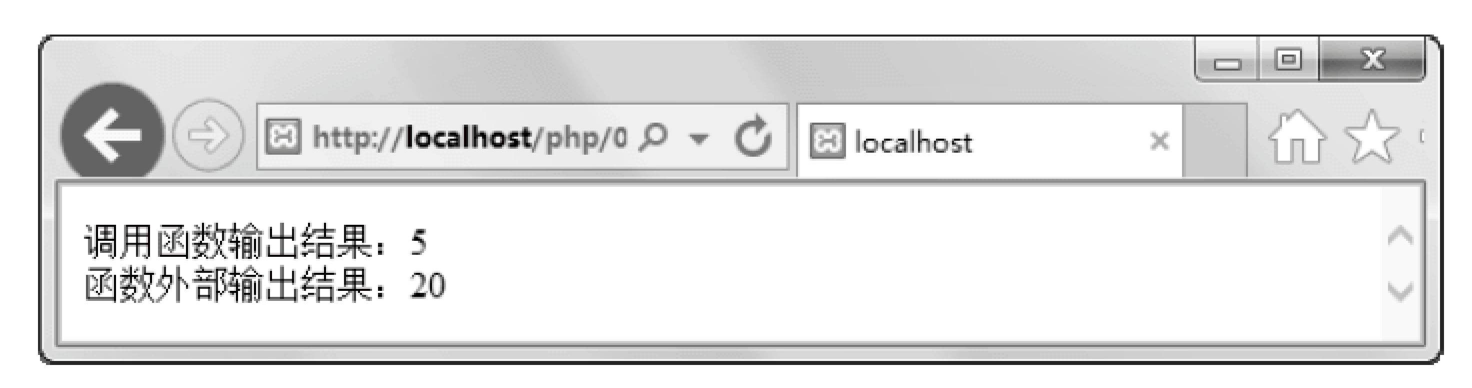


图 4.6　运行结果

在代码中我们可以看到函数内和函数外的变量是不同的，否则函数外部输出结果也将是 5 而不是 20。

【示例 4-7】以下代码试图输出函数中的变量。

```
<?php
    function print_num(){        //定义函数
```

```
        $x=6;                          //在函数中定义变量
    }
    print_num();                       //调用函数
    echo $x;
?>
```

代码运行结果如图 4.7 所示。



图 4.7　运行结果

代码在运行后输出了一个变量为定义的提示，这就证明了在函数外部不能够读取到函数内部定义的变量。

1. 全局变量

如果需要在函数中使用函数外的变量，那么就需要使用 global 来声明一个全局变量，它的一般形式如下：

```
global 变量 1,变量 2,变量 3…
```

【示例 4-8】以下代码演示使用 global 关键字来使用函数外部变量。

```
<?php
    function test(){              //定义函数
        global $a;                //声明全局变量
        echo $a;
    }
    $a=7;
    test();                       //调用函数
?>
```

代码运行结果如图 4.8 所示。



图 4.8　运行结果

从运行结果可以看到，虽然在函数中并没有定义变量 a，但是调用函数时仍然输出了变量 a 的值，这就是 global 定义的作用。

2. 静态变量

变量作用域的另一个特点就是静态变量。静态变量可以使函数中的变量状态保留而不会被销毁。首先来看一段不使用静态变量的示例。

【示例 4-9】以下代码演示调用不使用静态变量的函数。

```
<?php
```

```
    function test(){      //定义函数
        $a=0;             //定义变量
        $a++;             //变量递增
        echo $a;          //输出变量值
    }
    test();               //第一次调用函数
    test();               //第二次调用函数
    test();               //第三次调用函数
?>
```

代码运行结果如图 4.9 所示。

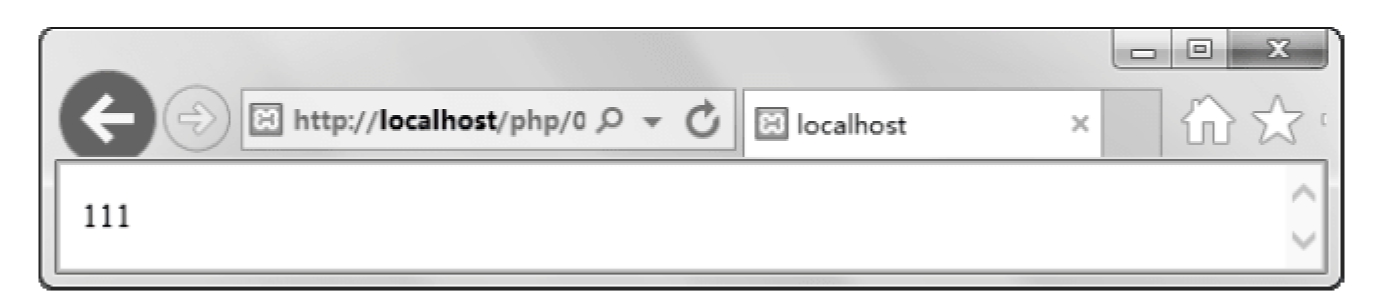


图 4.9　运行结果

以上输出结果证明了 3 次调用函数均输出 1，也就是说，函数中的变量 a 的值，在每次调用中均会被初始化。

静态变量使用 static 关键字定义，形式如下：

```
static 变量名=初始值;
```

【示例 4-10】以下代码演示使用 static 定义静态变量。

```
<?php
    function test(){      //定义函数
        static $a=0;      //定义静态变量
        $a++;             //变量递增
        echo $a;          //输出变量值
    }
    test();               //第 1 次调用函数
    test();               //第 2 次调用函数
    test();               //第 3 次调用函数
?>
```

代码运行结果如图 4.10 所示。



图 4.10　运行结果

以上代码仅在示例 4-9 的基础上定义了静态变量，但是输出结果却是不同的，这就是因为静态变量的状态会被保留，因而可以实现递增。

4.3　函数的其他使用方法

变量除了上一小节中讲解的常用使用方法之外，还有一些其他的使用方法，包括函数的递归调用、可变函数、匿名函数和函数的引用返回。下面我们就来学习这些使用方法。

4.3.1　可变函数

可变函数是指以一个变量作为函数名来调用函数。因此同样的形式会随着变量值的改变而调用不同的函数，常用的形式如下：

```
$variable()
```

【示例 4-11】以下代码演示可变函数的使用方法。

```
<?php
    function hello(){             //定义函数
        echo '<br />Hello!<br />';
    }
    function hellophp(){          //定义函数
        echo '<br />Hello PHP!<br />';
    }
    function good(){              //定义函数
        echo '<br />PHP is very good!<br />';
    }
    $func='good';                 //定义变量 func
    $func2='hello';               //定义变量 func2
    $func();                      //变量作为函数名
    $func2();                     //变量作为函数名
    $func='hellophp';             //对变量 func 重新赋值
    $func();                      //变量作为函数名
?>
```

代码运行结果如图 4.11 所示。

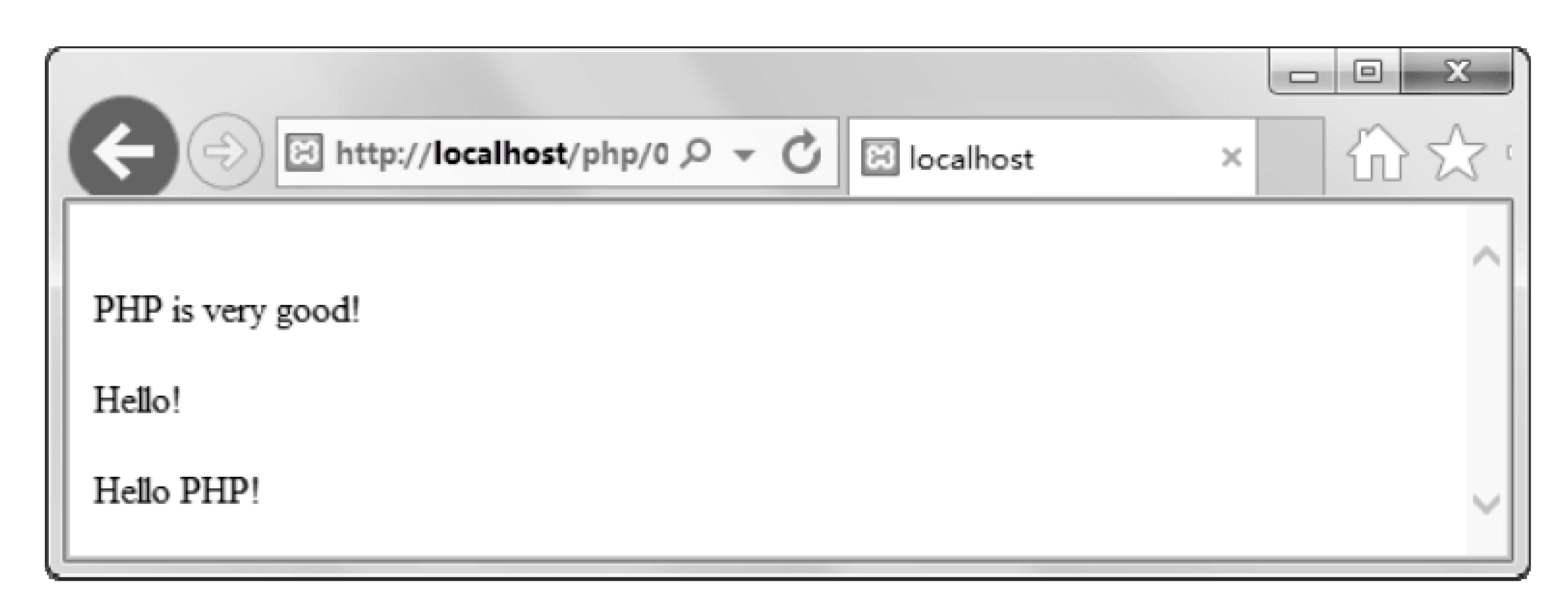


图 4.11　运行结果

4.3.2　函数的引用返回

函数的引用返回常用于对函数内的变量与函数外的变量建立联系，因而可以通过读取或者修改函数外变量的值来读取或者修改函数内的变量值。引用返回的函数形式如下：

```
function &函数名(参数列表){
    函数体;
    return 返回值;
}
```

【示例 4-12】以下代码演示函数引用返回的使用。

```
<?php
    function &test(){             //定义函数
```

```
        static $a=0;                //定义变量
        $a++;                       //变量递增
        return $a;                  //返回变量值
    }
    $quote=&test();                 //建立联系
    echo "\$a=$quote";              //输出函数中变量 a 的值
    test();                         //调用函数
    test();                         //调用函数
    echo "<br />\$a=$quote";        //输出函数中变量 a 的值
    $quote=15;                      //改变返回值$a 的值
    test();                         //调用函数
    echo "<br />\$a=$quote";        //输出函数中变量 a 的值
?>
```

代码运行结果如图 4.12 所示。

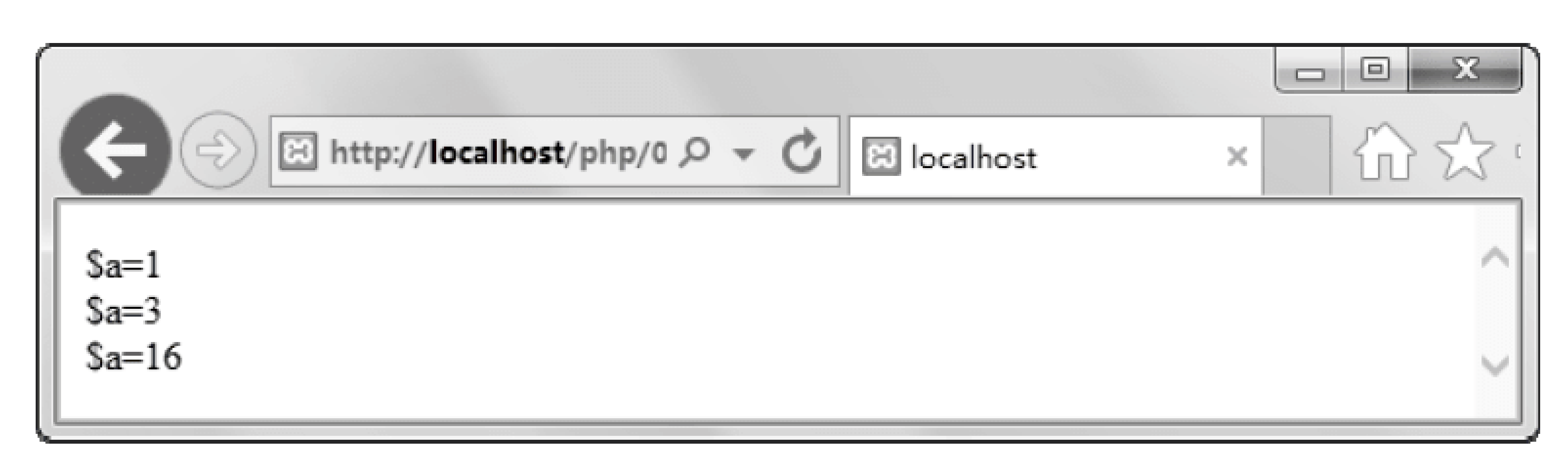


图 4.12　运行结果

从运行结果可以看到，函数在外部就实现了静态变量 a 从 3～16 的跨越，却只需要进行一次赋值而不使用引用返回，想要达到同样的结果就需要多次重复调用函数。

4.3.3　函数的递归调用

函数的递归调用即在函数体内调用函数自身。通常的形式如下：

```
function func_name(…){
    …;
    func_name(…);
    …;
}
```

下面就通过一个简单的题目来学习本节的知识，题目如下：初始情况下有一个细胞，它每 2 分钟会分裂出 1 个细胞。新生细胞有 1 分钟的成熟期，成熟后的细胞每 2 分钟会分裂出一个细胞。我们就来设计程序计算 15 分钟后共有多少细胞。

【示例 4-13】以下代码演示使用递归计算以上题目。

```
<?php
    function cell($start,$end){          //定义细胞函数，start 为开始
                                           时间，end 为结束时间
        global $sum;                     $start=$i+2
        for($i=$start+1;$i<$end;$i=$i+2) $end=$end
            if($i+2<=$end){
                cell($i+2,$end);         //递归调用
                $sum=$sum+1;
            }
    }
    $sum=1;
    cell(-1,15);     //由于开始时是有一个成熟的细胞，因此需要将时间提前 1 分钟
```

```
    echo "15 分钟后为$sum 个细胞。";
?>
```

代码运行结果如图 4.13 所示。

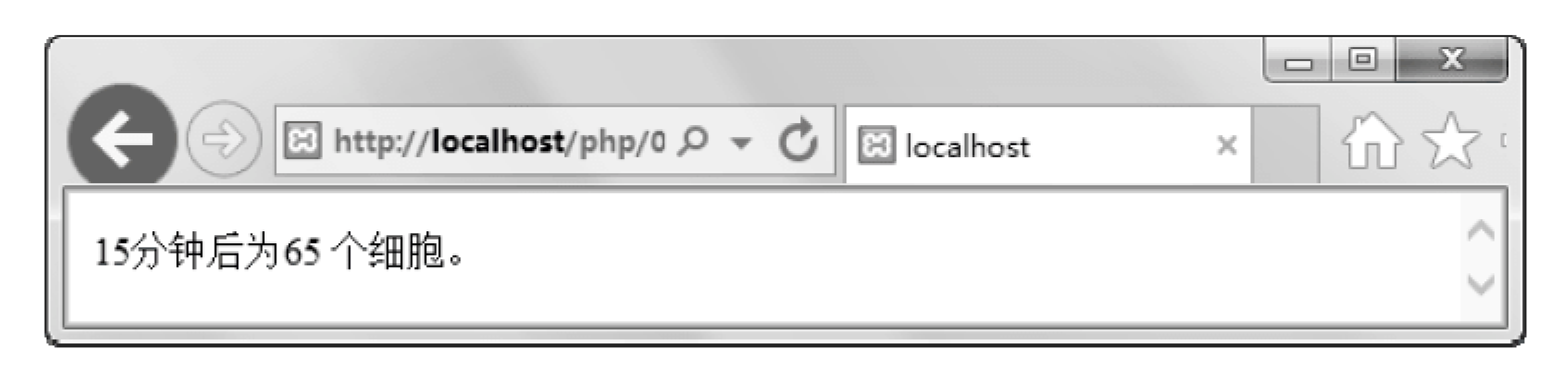


图 4.13　运行结果

递归调用是比较耗费系统资源的，虽然 15 分钟的细胞数可以很容易地计算出来，但是计算一个比较长的时间浏览器，通常会响应比较长的时间。

4.3.4　匿名函数

匿名函数也称为闭包函数，它是一个没有函数名的函数，通常会作为回调函数的参数，它的形式如下：

```
function (参数列表){
    语句;
}
```

匿名函数也可以与一个变量绑定，常用形式如下：

```
$varable = function (参数列表){
    语句;
}
```

匿名函数作为回调函数参数的知识我们不在这里讲解，下面我们就以匿名函数与变量绑定的形式来学习。

【示例 4-14】以下代码演示匿名函数与变量绑定。

```
<?php
    $func=function ($x,$y){                //匿名函数与变量绑定
        return $x+$y;
    };
    echo '5+6='.$func(5,6);                //使用匿名函数
    echo '<br />15+16='.$func(15,16);
?>
```

代码运行结果如图 4.14 所示。

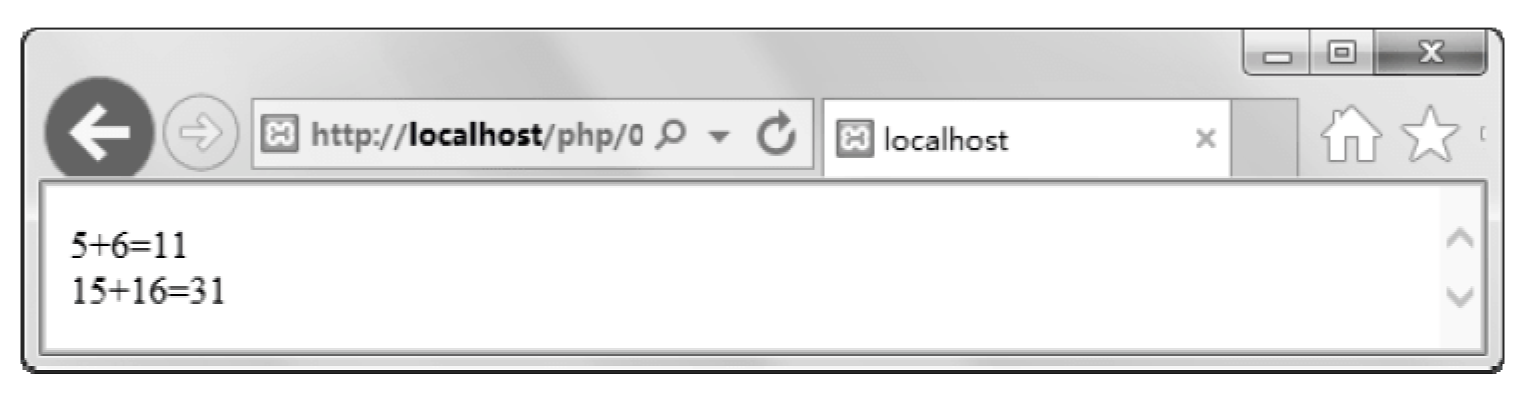


图 4.14　运行结果

在将匿名函数与变量绑定的时候，需要注意在匿名函数结尾需要以分号结束，这类似于为变量赋值。

4.4　小　　结

本章主要讲解了 PHP 函数部分的知识，包括使用函数的优势，以及详细讲解了函数的各种使用方法，让读者使用函数时候基本没有盲点。函数的递归调用是初学者比较难掌握的知识，读者现在无须完全理解递归，递归的知识将在不断的学习和实践中会逐渐深入理解。

4.5　本 章 习 题

1. 定义一个函数，并且 3 次调用使其输出 3 次“Hello!”。执行的效果应如图 4.15 所示。

2. 定义一个函数，使其每次被调用均会输出传入的字符串参数。执行的效果应如图 4.16 所示。

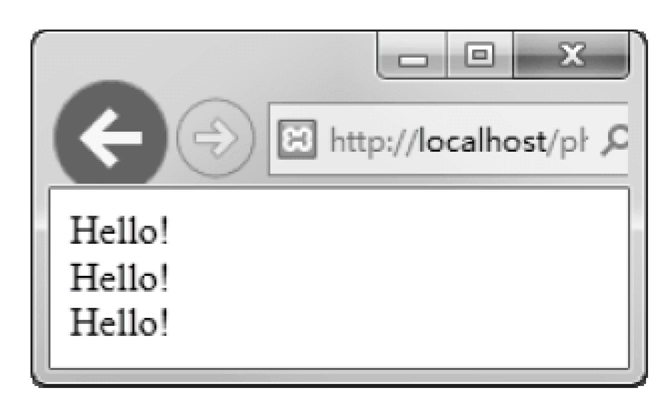


图 4.15　运行效果



图 4.16　运行效果

3. 定义三个函数 operation()、add()和 sub()，operation()函数根据传入的运算符对两个操作数进行相应的运算。运行的效果应如图 4.17 所示。

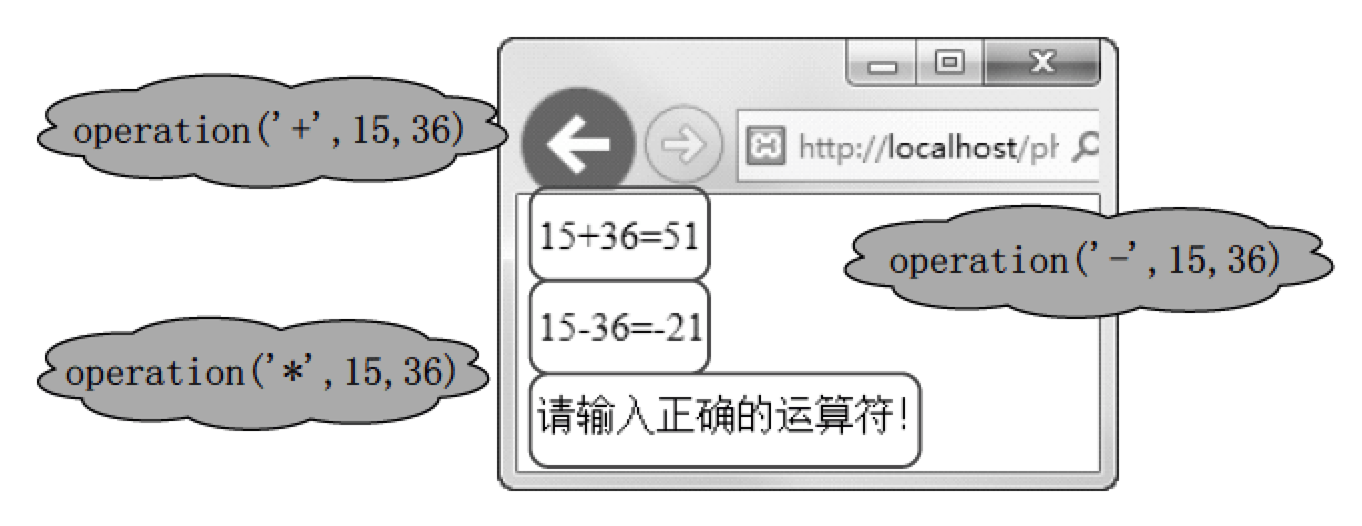


图 4.17　运行效果

4. 使用递归调用实现计算表 4.1 所示的第 N 项值。

表 4.1　斐波那契数列

项	1	2	3	4	5	6	…	N
值	1	1	2	3	5	8	…	(N-1)+(N-2)

第 5 章　数　　组

数组是将多个数据集合在一起的一个形式。它在 PHP 中是非常强大的一种数据类型，可以用来存储多种类型的多个数据。本章将首先介绍数组的优势，然后循序渐进地从简单创建一个数组、修改数组中的数据到常用的遍历数组再到灵活操作数组中指针的形式来学习，从而让读者很好地掌握 PHP 的数组。

5.1　使 用 数 组

数组在一些方面有其独特的优势。本节将介绍数组使用方法的基础知识，以及关联数组和索引数组的使用和应用方面，从而让读者对如何正确使用数组有一个明确的了解。

5.1.1　使用数组的优势

数组的优势在于可以集中处理大批量的数据，而不需要每次更换变量名。由于是集中处理数据，所以处理方式上有很多优化和改进。PHP 中提供了专门的数组运算符及非常多的操作数组的函数，因此 PHP 中的数组使用可以非常灵活。当然灵活性的提高是建立在不同细节上的，因此我们就需要从细节讲起，让读者了解数组的每一个细节，然后根据需要来灵活操作数组。

我们不妨来设想下面这个情景：如果要求你统计一个班级中成绩的平均分，那么我们必须要声明等于学生个数的变量来存储每个人的成绩，然后再将所有成绩相加后再取得平均分简略的代码可以写成如下的形式：

```
<?php
    $stu01=67;                                      //声明多个变量存储学生成绩
    $stu02=77;
    $stu03=96;
    ...
    $stun=100;
    $average=($stu01+$stu02+stu03+...+$stun)/n;    //求取平均值
?>
```

我们可以看到，这种形式需要声明很多变量来存储学生的成绩，而使用数组实现同样的功能，我们可以把程序改为如下形式：

```
<?php
    $stu=array(67,77,96,...,100);       //定义一个数组
    $total=array_sum($stu);             //求取总成绩
    $average=$total/n;                  //求取平均值
?>
```

使用数组实现相同功能的代码精简度是显而易见的，当然这里只是作为一个演示，读者此刻无须理解代码中使用的函数。

5.1.2　数组使用基础

在 5.1.1 节中我们已经了解了数组所具备的优势，本节我们就来讲解使用数组的基础知识，主要包括定义数组、访问数组元素、添加/修改数组元素和删除数组元素的知识。

1. 定义数组

PHP 中定义数组使用的是 array 结构，语法形式如下：

```
array array ([mixed values ]… )
```

这是定义数组的最基本形式，array 的参数被称元素可以是 0 个或者多个不同类型的数据，每个参数为“索引=>值”的键值对形式。如果索引被省略，则由系统自动添加从 0 开始的整数索引，这种由系统指定索引的函数被称为索引数组。数组中元素的个数被称为数组的长度，它会随着元素的增减而变化。下面就来定义一个数组：

```
$arr=array(98,’hello’,67,’A’,85,NULL);
```

上面的代码中$arr 是数组的名称，用来保存定义的数组。该数组的长度为 6。

2. 访问数组元素

访问数组中的元素是通过数组名指定索引来完成的，语法形式如下：

```
数组名[索引]
```

语法中的索引被放在方括号中，可以为变量、常量或者表达式。

【示例 5-1】以下代码演示访问数组中的元素。

```
<?php
    $arr=array(98,'hello',67,'A',85,NULL);      //定义一个数组
    echo "输出一个元素：{$arr[0]}";              //输出数组的元素
    echo "<br />输出第二个元素：{$arr[1]}";
    echo "<br />输出第三个元素：{$arr[2]}";
    echo "<br />输出第四个元素：{$arr[3]}";
    echo "<br />输出第五个元素：{$arr[4]}";
    echo "<br />输出第六个元素：{$arr[5]}";
?>
```

注意：数组的默认索引是从 0 开始的。在输出语句中使用花括号可以输出变量的值而不是以字符串输出。

代码运行结果如图 5.1 所示。

从运行结果可以看到，代码正确访问到了数组中的元素。

我们还可以把一个变量作为数组的下标来访问数组的元素，形式如下：

```
数组名[变量名]
```

【示例 5-2】以下代码演示使用变量作为数组下标来访问数组中的元素。

```
<?php
    $arr=array(98,'hello',67,'A',85,NULL);     //定义一个数组
    $x=0;                                       //定义三个作为下标的变量
    $y=3;
    $z=5;
    echo "下标为{$x}的元素为{$arr[$x]}。<br />";  //输出对应下标及元素值
    echo "下标为{$y}的元素为{$arr[$y]}。<br />";
    echo "下标为{$z}的元素为{$arr[$z]}。<br />";
    $x=2;                                       //为变量 x 重新赋值
    echo "下标为{$x}的元素为{$arr[$x]}。";        //输出对应下标及元素值
?>
```

代码运行结果如图 5.2 所示。

从运行结果可以看到，我们可以采用这种方式来正确输出数组对应的元素。我们还可以将一个表达式作为数组的下标来访问数组的元素。

【示例 5-3】以下代码演示使用表达式作为数组下标来访问数组中的元素。

```
<?php
    function num(){                                   //定义函数
        $x=7;
        $y=6;
        return $x-$y;                                 //返回值为 7
    }
    $arr=array(98,'hello',67,'A',85,NULL);            //定义一个数组
    $a=1;                                             //定义两个变量
    $b=3;
    echo '数组下标为 9 的元素为：'.$arr[$a+$b];         //表达式作为下标
    echo '<br />数组下标为 7 的元素为：'.$arr[num()];   //表达式作为下标
?>
```

代码运行结果如图 5.3 所示。



图 5.1　运行结果



图 5.2　运行结果



图 5.3　运行结果

以上代码中我们只使用了简单的示例来讲解，读者可以在实际使用时候融汇贯通。

5.2　数组常用操作

本节将要学习的是常用的数组操作，主要包括遍历、比较、合并、分割、排序和过滤数组元素。在学习这些知识之前，我们首先需要认识三个函数 count、print_r 和 var_dump，它们的原型如下所示：

```
int count ( mixed $var [, int $mode ] )
bool print_r ( mixed $expression [, bool $return ] )
void var_dump ( mixed $expression [, mixed $expression [, $... ]] )
```

count 函数用来计算数组中元素的个数或者对象中属性的个数，这里我们只用来计算数组元素的个数。print_r 函数用来打印变量的详细信息。var_dump 函数用来打印一个或多个表达式的结构信息，包括表达式的类型与值。

5.2.1　for 循环遍历数组

遍历数组即依次对数组中的每个元素进行访问且仅访问一次。我们知道默认数组的索引值是从 0 开始递增的，而且可以将变量作为索引来访问数组的元素。因此，我们完全可以使用循环递增一个作为索引的变量来遍历一个数组。

【示例 5-4】以下代码演示使用 for 循环遍历输出一个数组中的元素。

```
<?php
    $arr=array(63,'abc',45,'hello',3,7,9,'DEF');    //定义一个索引数组
    echo '在数组$arr 中：<br />';
    for($i=0;$i<count($arr);$i++){        //将循环控制变量作为访问数组的下标
        echo "第{$i}个元素：{$arr[$i]}<br />";        //输出数组元素
    }
?>
```

代码运行结果如图 5.4 所示。



图 5.4　运行结果

结合代码我们可以看到，通过简单的 for 循环语句遍历输出了数组的所有元素。

5.2.2　合并数组

合并数组即为将两个或者多个数组合并为一个数组。合并数组有对应的运算符和多个函数可以完成，它们的区别就在于使用不同的规则对数组进行合并。

1. 使用联合运算符合并数组

PHP 中有专门的数组运算符，用来合并数组的运算符称为联合运算符，它的符号与加法的符号相同。使用的形式如下：

```
$arr = $arr1 + $arr2
```

它会将右操作数中与左操作数中相同索引的元素去除后，将剩余元素加在左操作数后而不会发生覆盖。

【示例 5-5】以下代码演示数组联合运算符（+）的使用。

```
<?php
```

```
    $arr1=array('a','b','c');                  //定义一个数组
    echo '数组$arr1 的信息：<br />';
    print_r($arr1);                            //输出数组信息
    $arr2=array('d','e','f','g');              //定义一个数组
    echo '<br />数组$arr2 的信息：<br />';
    print_r($arr2);                            //输出数组信息
    $arr=$arr1+$arr2;                          //联合数组
    echo '<br />联合后的数组$arr 的信息：<br />';
    print_r($arr);                             //输出联合后的数组信息
?>
```

代码运行结果如图 5.5 所示。

从以上输出结果就可以看到，第一个数组中的元素并不会被覆盖。

2. 使用系统函数合并数组

基本的合并数组函数是 array_merge()函数，它可以将一个或者多个数组合并起来，将后一个数组的元素加在前一个数组的末尾。函数原型如下：

```
array array_merge ( array $array1 [, array $array2 [, array $... ]] )
```

参数 array1、array2…为将要合并的数组，该函数的特点是可以一次合并多个数组。

【示例 5-6】以下代码演示使用 array_merge()函数合并多个数组。

```
<?php
    $arr1=array('a','b','c');                  //定义一个数组
    echo '数组$arr1 的信息：<br />';
    print_r($arr1);                            //输出数组信息
    $arr2=array('d','e','f','g');              //定义一个数组
    echo '<br />数组$arr2 的信息：<br />';
    print_r($arr2);                            //输出数组信息
    $arr3=array('e','g','h');                  //定义一个数组
    echo '<br />数组$arr3 的信息：<br />';
    print_r($arr3);                            //输出数组信息
    $arr=array_merge($arr1,$arr2);             //合并 2 个数组
    echo '<br />合并$arr1 和$arr2 后的数组$arr 信息：<br />';
    print_r($arr);                             //输出联合后的数组信息
    $arr=array_merge($arr1,$arr2,$arr3);       //合并 3 个数组
    echo '<br />合并$arr1、$arr2 和$arr3 后的数组$arr 信息：<br />';
    print_r($arr);                             //输出联合后的数组信息
?>
```

代码运行结果如图 5.6 所示。

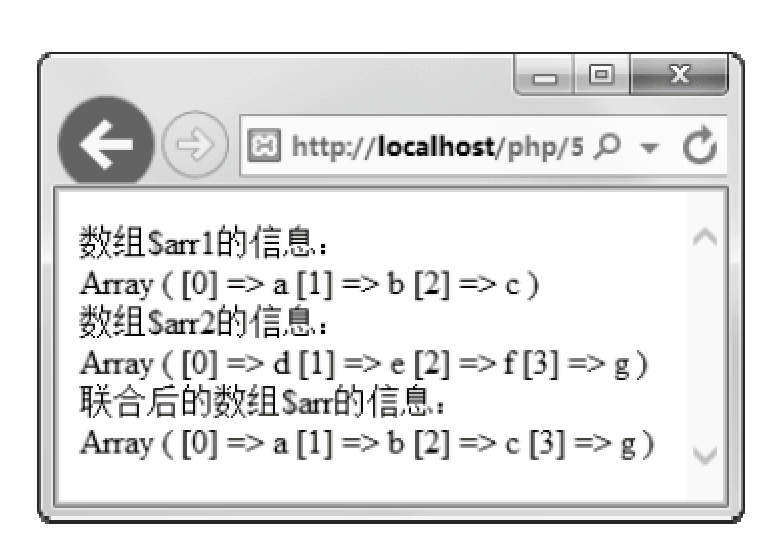


图 5.5　运行结果

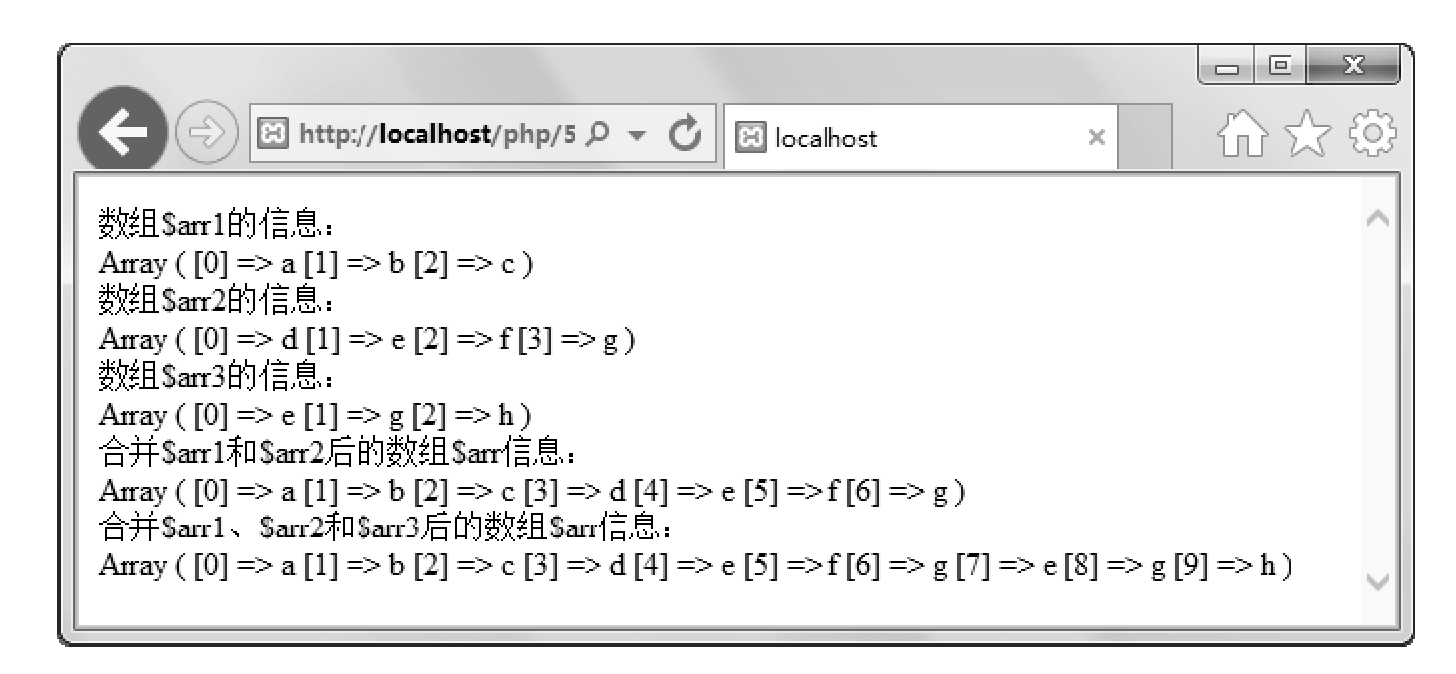


图 5.6　运行结果

在上面的代码中分别使用 array_merge 函数合并两个和三个数组，并输出各个数组的详细信息。这里需要读者了解的是该函数不会覆盖数组的元素。

5.2.3　获取数组的交集和差集

获取数组的交集即为获取指定数组与其他数组中相同的元素；获取数组的差集即为获取指定数组与其他数组中不同的元素。在 PHP 中提供了获取交集和差集的多个函数，这里只讲解最主要的两个，它们的函数原型如下：

```
array array_intersect (array $array1 , array $array2 [, array $ ... ])
                                            //获取数组的交集
array array_diff ( array $array1 , array $array2 [, array $ ... ] )
                                            //获取数组的差集
```

这两个函数均可接受多个参数，而且均会返回函数中参数 array1 与其他数组的交集（差集）。这里我们也可借用数学中的题目来学习这个函数。例如有两个数组$arr1 和$arr2 用来保存两组参加知识竞赛学生的学号。前提是一个学生可以参加多项竞赛，那么我们就可以用以上函数来取得两项竞赛都参加的学生（交集），同样也可以取得一组中只参加一项竞赛的学生。

【示例 5-7】以下代码演示使用 array_intersect()函数和 array_diff()函数，获取学生数组的交集和差集并输出。

```
<?php
    $arr1=array(1,3,4,5,34,78,99);      //参加第一项竞赛的学生学号数组
    $arr2=array(5,6,7,3,56,34,8,9);     //参加第二项竞赛的学生学号数组
    $arr=array_intersect($arr1,$arr2); //获取两个数组的交集
    echo '两项竞赛均参加的学生学号有：<br />';
    print_r($arr);                      //输出数组的详细信息
    $arr=array_diff($arr1,$arr2);       //获取相对数组$arr1 的差集
    echo '<br />$arr1 中只参加一项竞赛的学生学号有：<br />';
    print_r($arr);                      //输出数组的详细信息
    $arr=array_diff($arr2,$arr1);       //获取相对数组$arr2 的差集
    echo '<br />$arr2 中只参加一项竞赛的学生学号有：<br />';
    print_r($arr);                      //输出数组的详细信息
?>
```

代码运行结果如图 5.7 所示。

与数学中只能获取数值的交集与差集不同的是，PHP 中可以使用以上两个函数来获取其他类型元素的交集和差集。

【示例 5-8】以下代码演示使用 array_intersect()函数和 array_diff()函数获取其他类型数组的交集和差集。

```
<?php
    $arr1=array('A','B','C','D','E');      //定义两个数组
    $arr2=array(1,'A',2,'B',3,'C');
    $arr=array_intersect($arr1,$arr2);     //获取数组的交集
    echo '数组$arr1 和数组$arr2 的交集是：<br />';
    print_r($arr);
    $arr=array_diff($arr1,$arr2);          //获取$arr1 与$arr2 的差集
    echo '<br />数组$arr1 和数组$arr2 的差集是：<br />';
    print_r($arr);
```

```
    $arr=array_diff($arr2,$arr1);          //获取$arr2 与$arr1 的差集
    echo '<br />数组$arr2 和数组$arr1 的差集是：<br />';
    print_r($arr);
?>
```

代码运行结果如图 5.8 所示。



图 5.7　运行结果



图 5.8　运行结果

使用 array_intersect()函数和 array_diff()函数获取多个数组的交集和差集，与获取两个数组的交集和差集类似，这里就不再详细讲解。

5.2.4　数值元素相关计算

在使用数组的过程中，很多时候需要对数组中的值进行求值运算。在本章开头的代码中，就使用了 array_num 来计算数组中元素的和。除了计算和的函数外还提供了 array_product()函数来计算数组中元素的乘积。本节我们就来介绍这两个系统函数。

1. array_sum()函数

array_sum()函数用来计算数组中所有值的和，函数原型如下：

```
number array_sum ( array $array )
```

参数 array 即为要求和的数组。

注意：array_sum()函数会将数字字符串转换为数值，将布尔值 TRUE 转换为 1，而把其他不能转换为数值的通常转换为 0 来计算。

【示例 5-9】以下代码演示使用 array_sum()函数计算数组中元素的值。

```
<?php
    $arr1=array(1,2,3,4,'5','05',TRUE);//等价于 1+2+3+4+5+5+1=21
    $arr2=array(1,2,'ABC',3,'hello',4,'5','05',NULL);
                                       //等价于 1+2+0+3+0+4+5+5+0=20
    echo '$arr1 中元素值的和为：'.array_sum($arr1);
                                       //求取数组$arr1 中数值的和
    echo '<br />$arr2 中元素值的和为：'.array_sum($arr2);
                                       //求取数组$arr2 中数值的和
?>
```

代码运行结果如图 5.9 所示。

2. array_product()函数

array_product()函数用来计算数组中所有值的乘积，函数原型如下：

```
number array_product ( array $array )
```

参数 array 即为要求乘积的数组。

注意：与 array_sum()函数类似，array_product()函数会将数字字符串转换为数值，将布尔值 TRUE 转换为 1，而把其他不能转换为数值的通常转换为 0 来计算。因此对于含有不能转换为数值元素的数组，使用该函数是无实际意义的，因为任何值与 0 的乘积都为 0。

【示例 5-10】以下代码演示使用 array_product()函数计算数组中元素的乘积。

```
01  <?php
02      $arr1=array(3,4,5,6,'7',TRUE);       //等价于 3*4*5*6*7*1=2520
03      $arr2=array(3,4,5,6,'7','hello');    //等价于 3*4*5*6*7*1*0=0
04      echo '计算数组$arr1 中元素的乘积为：'.array_product($arr1);
                                            //对数组元素求乘积
05      echo '<br />计算数组$arr2 中元素的乘积为：'.array_product($arr2);
                                            //对数组元素求乘积
06  ?>
```

代码运行结果如图 5.10 所示。



图 5.9　运行结果



图 5.10　运行结果

从运行结果可以看到，代码第 5 行求取数组$arr2 中元素的乘积得到的结果为 0，这是因为在代码第 3 行中的数组元素 hello 会被转换为 0 导致的，因此我们实际应用中应该尽量避免这种情况。

5.3　增加与删除数组元素

在 PHP 中，数组定义后，通常不是一成不变的。在程序执行过程中，数组的元素可能被增加、删除和修改，这也是 PHP 的数组之所以非常强大的一个原因。下面就来详细介绍这些知识。

5.3.1　添加/修改数组元素

在有些情况下，数组中的元素并不是在定义数组时候就可以确定的，通常会在程序运行时动态地向数组中增加元素。为数组添加元素的语法类似于赋值，如下所示。

```
数组名[索引] = 值
```

为数组增加元素需要使用未在数组中使用的索引或者不指定索引，不指定索引添加的数组索引同样由系统指定，指定规则为获取当前数组中最大整数索引值，并在其基础上加 1 作为新元素的索引。

在接下来的示例中将用到 rand()函数，它用来生成随机数，函数原型如下：

```
int rand ([ int $min ], int $max )
```

参数 min 和 max 分别用来指定生成的随机数的最小和最大范围。

【示例 5-11】以下代码演示为数组添加元素。

```
01  <?php
02      $arr=array();                   //定义一个数组，它没有任何元素
03      echo '增加元素之前数组中元素的个数为：'.count($arr);      //输出数组个数
04      for($i=0;$i<3;$i++)
05          $arr[]=rand(0,99);          //使用 for 循环为数组循环添加 3 个元素，
                                          rand 函数用啦产生随机数
06      echo '<br />增加元素之后数组中的元素个数为：'.count($arr);
07      echo '<br />输出数组的详细信息：<br />';
08      print_r($arr);                  //输出数组详细信息
09      $arr[8]=100;                    //为数组添加指定索引的元素
10      for($i=0;$i<3;$i++)
11          $arr[]=rand(0,99);          //使用 for 循环为数组循环添加 3 个元素，
                                          rand 函数用啦产生随机数
12      echo '<br />增加元素之后数组中的元素个数为：'.count($arr);
13      echo '<br />输出数组的详细信息：<br />';
14      print_r($arr);                  //输出数组详细信息
15  ?>
```

代码运行结果如图 5.11 所示。

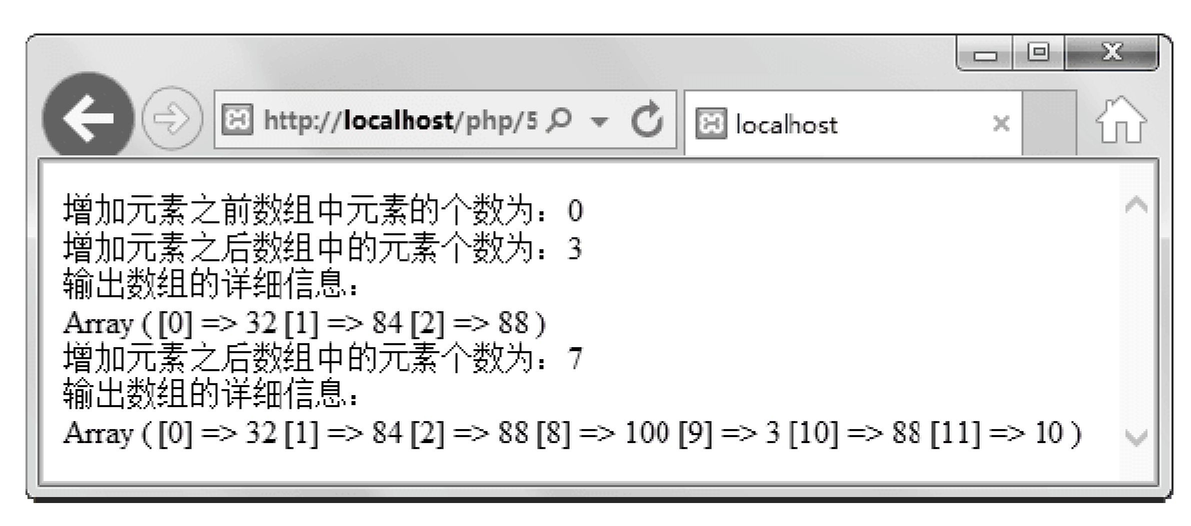


图 5.11　运行结果

本段代码需要读者理解代码中指定索引和不指定索引的形式下，元素索引的变化，当代码第 9 行为键指定为 8 的时候，第 11 行添加的元素的键就为会以 8 作为基础递增。

修改数组中的元素只要访问到指定的元素，然后为其重新赋值即可，语法形式如下：

```
数组名[索引] = 值
```

【示例 5-12】以下代码演示修改数组中的元素。

```
<?php
    $arr=array(98,'hello',67,'A',85,NULL); //定义一个数组
    echo '输出数组修改元素之前的详细信息：<br />';
    print_r($arr);                         //输出数组修改之前的详细信息
    $arr[1]='Hi';                          //修改数组的元素
    $arr[5]='good!';
    $arr[3]=100;
    echo '<br />输出数组修改元素之后的详细信息：<br />';
    print_r($arr);                         //输出数组修改之前的详细信息
?>
```

代码运行结果如图 5.12 所示。

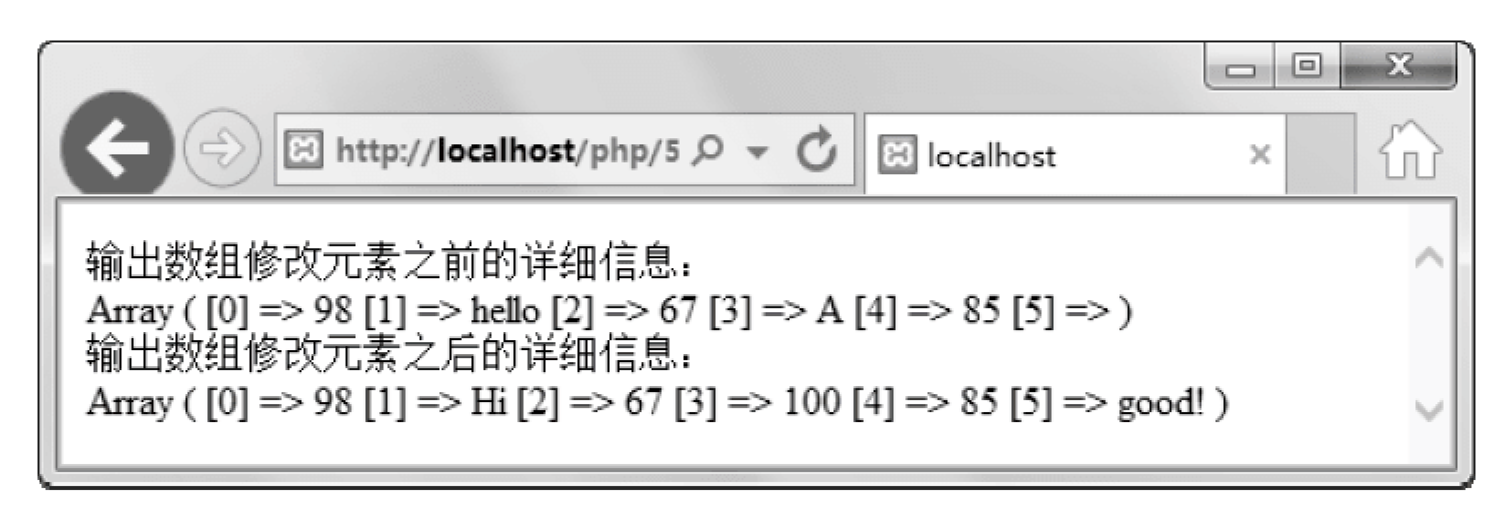


图 5.12　运行结果

从输出结果可以明确看到数组被正确修改。

5.3.2　删除数组/数组中的元素

当数组中的元素不再需要的时候我们就可以删除它。同样地，当整个数组不再需要的时候我们同样可以删除整个数组。完成这些操作可以使用的函数是 unset()，它用来释放指定的变量，函数原型如下：

```
void unset ( mixed $var [, mixed $var [, $... ]] )
```

unset 可以接受多个参数，参数 var 即为需要释放的变量，在本章学习中我们用来释放数组中的元素和数组。

【示例 5-13】以下代码演示使用 unset()函数释放数组元素和数组。

```
01  <?php
02      $arr=array(98,'hello',67,'A',85,NULL);      //定义一个数组
03      echo '删除元素之前数组中的元素个数为：'.count($arr); //输出数组元素个数
04      echo '<br />数组的详细信息为：<br />';
05      print_r($arr);                              //输出数组详细信息
06      unset($arr[5]);                             //删除索引为 5 的元素
07      echo '<br />删除元素后前数组中的元素个数为：'.count($arr);
                                                    //输出数组元素个数
08      echo '<br />数组的详细信息为：<br />';
09      print_r($arr);                              //输出数组信息
10      unset($arr[1]);                             //删除索引为 1 的元素
11      echo '<br />删除元素后前数组中的元素个数为：'.count($arr);
                                                    //输出数组元素个数
12      echo '<br />数组的详细信息为：<br />';
13      print_r($arr);                              //输出数组详细信息
14      unset($arr);                                //删除整个数组
15      echo '<br />释放$arr 变量后尝试输出数组信息：<br />';
16      print_r($arr);                      //尝试数组数组信息（运行会报错）
17  ?>
```

代码运行结果如图 5.13 所示。

从运行结果中可以看到，指定的元素和数组都被删除。在第 14 行代码删除整个数组后，在第 16 行尝试访问数组会提示变量未定义。需要注意的是使用 unset()函数删除数组元素并不会影响数组的索引。

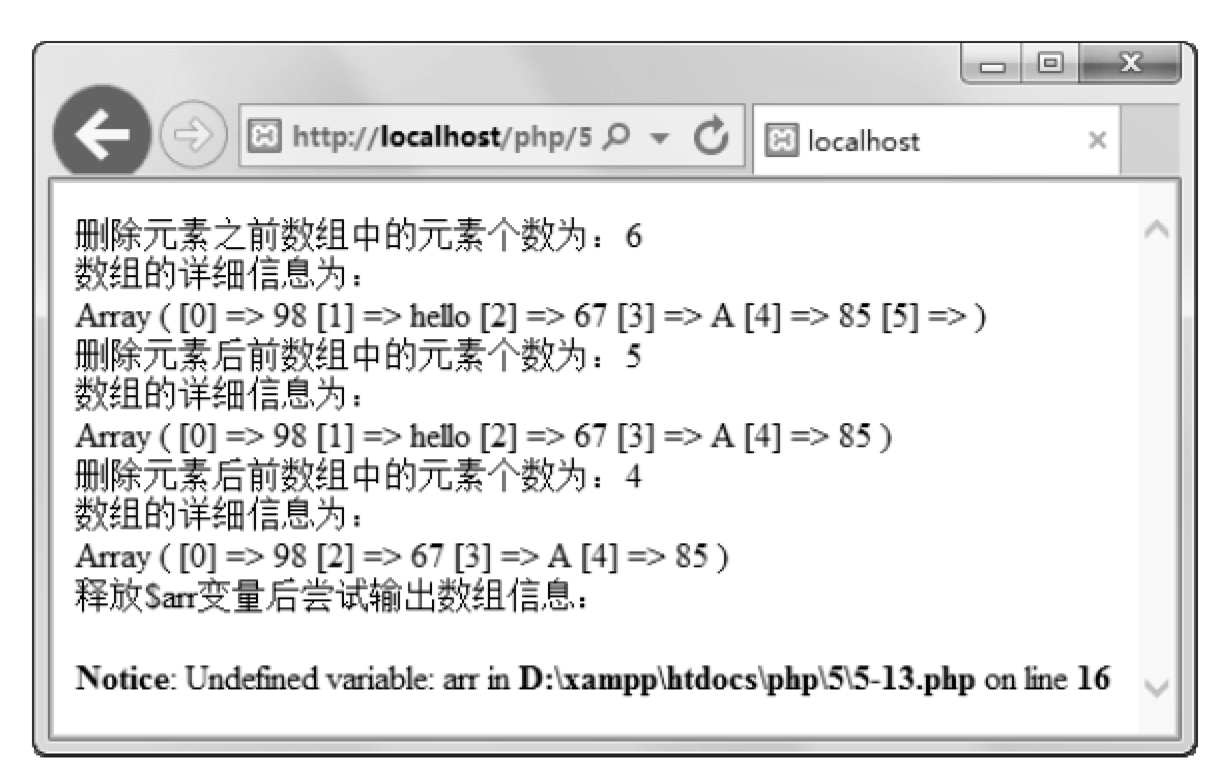


图 5.13　运行结果

5.4　遍 历 数 组

遍历数组即依次对数组中的每个元素进行访问且仅访问一次。由于遍历是依次访问，而且数组的索引又是依次递增的，因此使用常规的循环都可以完成对数组的遍历。一些常用的操作也是基于遍历来完成的，我们也在本节讲解。

5.4.1　排序数组

排序在数组中是比较常用的操作，例如可以用来完成根据学生的成绩进行排名等操作。首先我们来使用 for 循环对一个数组的元素进行排序。

【示例 5-14】以下代码演示使用 for 循环对一个数组中的元素进行从大到小的排序。

```
<?php
    $arr=array(35,43,56,2,76,23,47,55,71);        //定义一个数组
    echo '数组排序之前的信息：<br />';
    print_r($arr);                                //输出排序前的数组信息
        for($i=0;$i<count($arr);$i++){            //对数组进行排序
            for($j=0;$j<count($arr)-1;$j++){
                if($arr[$j]>$arr[$j+1]){          //判断前后元素的大小
                $b=$arr[$j];                      //交换元素的值
                $arr[$j]=$arr[$j+1];
                $arr[$j+1]=$b;
                }
            }
        }
    echo '<br />数组排序之后的信息：<br />';
    print_r($arr);                                //输出排序后的数组信息
?>
```

代码运行结果如图 5.14 所示。

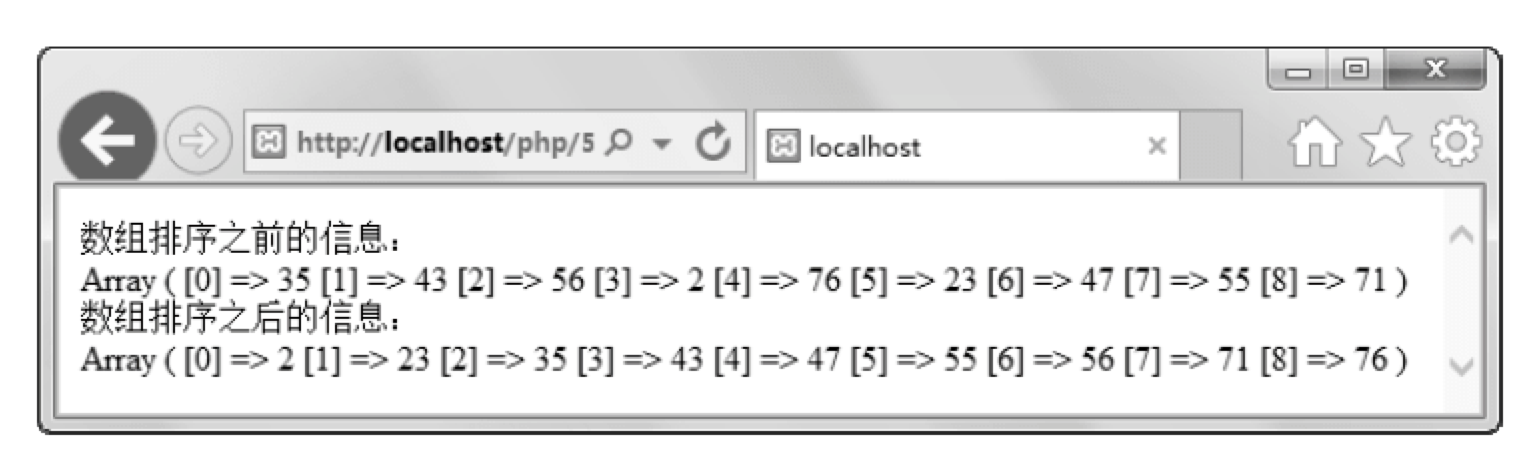


图 5.14　运行结果

从运行结果排序后的数组可以看出，我们完成了对数组的排序。由于排序的语句（代码中 5～13 行）是相对固定的，因此可以将其封装为函数来使用。

【示例 5-15】以下代码演示将示例 5-14 中完成排序的代码，封装为函数并对多个数组进行排序。

```
<?php
    function mysort($arr){        //将排序的代码封装为函数
        echo '<br />数组排序之前的信息：<br />';
        print_r($arr);            //输出排序前的数组信息
        for($i=0;$i<count($arr);$i++){       //对数组进行排序
            for($j=0;$j<count($arr)-1;$j++){
                if($arr[$j]>$arr[$j+1]){
                 $b=$arr[$j];
                 $arr[$j]=$arr[$j+1];
                 $arr[$j+1]=$b;
                }
            }
        }
        echo '<br />数组排序之后的信息：<br />';
        print_r($arr);                       //输出排序前的数组信息
    }
    $arr1=array(654,853,123,147,259,377);   //定义数组
    $arr2=array(23,45,1,12,22,90,145,75);
    $arr3=array(123,23,741,638,254,36,11);
    mysort($arr1);                           //使用自定义函数排序
    mysort($arr2);                           //使用自定义函数排序
    mysort($arr3);                           //使用自定义函数排序
?>
```

代码运行结果如图 5.15 所示。

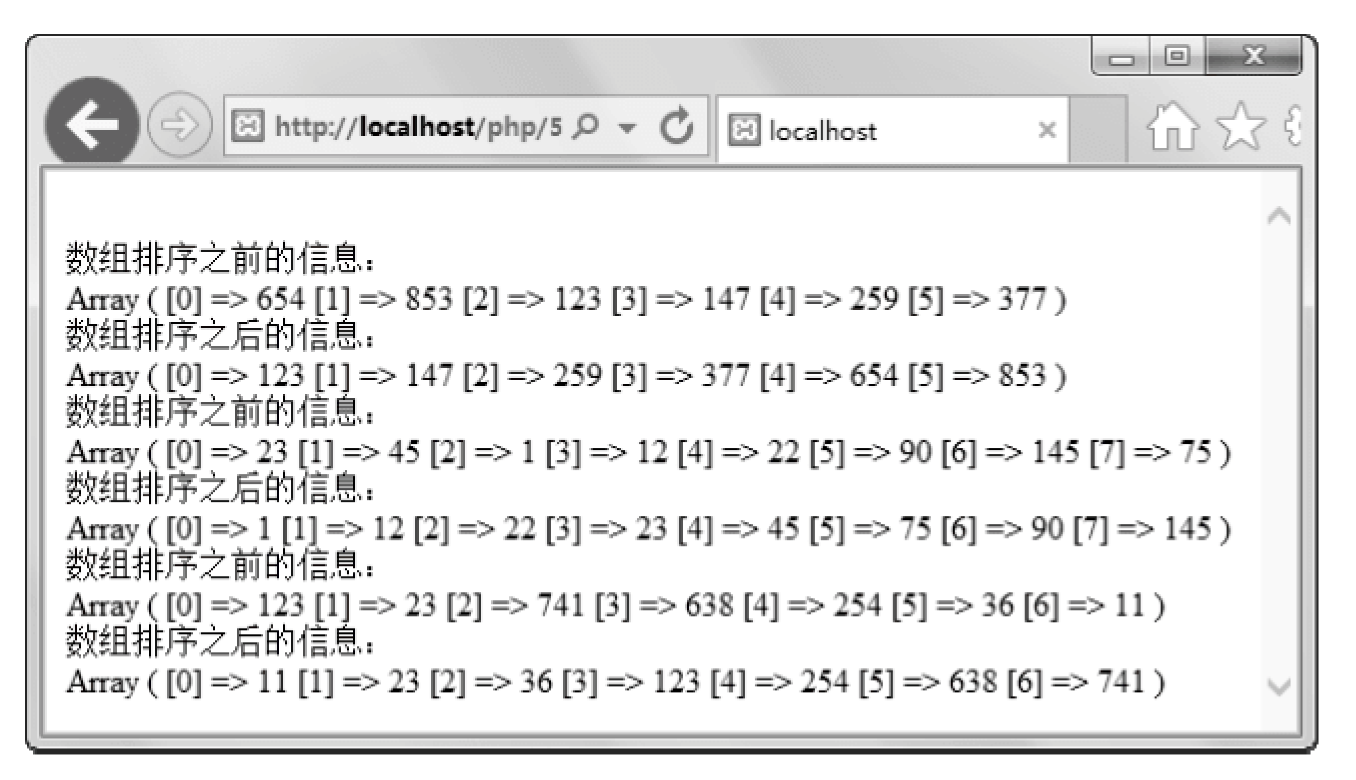


图 5.15　运行结果

从运行结果可以看到，使用我们的自定义函数很轻松就完成了对三个数组的排序，这也再一次体现了函数的优势。由于排序操作在数组中很常用，因此 PHP 提供了多个系统排序函数供我们使用，它们主要有 sort、rsort 和 usort。这 3 个函数的原型如下：

```
bool sort (array &$array [, int $sort_flags ]) //用于对数组正向（从小到大）排序
bool rsort (array &$array [, int $sort_flags ])//用于对数组逆向（从大到小）排序
```

这两个函数的使用方法类似，参数 array 为要进行排序的数组；参数 fort_flags 用来改变排序的方式，主要有如下 3 个选项：

- SORT_REGULAR：正常比较元素（默认方式）；
- SORT_NUMERIC：元素被作为数字来比较；
- SORT_STRING：元素被作为字符串来比较。

【示例 5-16】以下代码演示使用 sort()函数对数组进行排序。

```
<?php
    $arr=array(2,54,167,'a','A','12');  //定义一个数组
    echo '数组排序之前的信息：<br />';
    print_r($arr);                      //输出数组的信息
    echo '<br />数组普通排序之后的信息：<br />';
    sort($arr);                         //对数组进行排序
    print_r($arr);
    echo '<br />数组作为数字排序之后的信息：<br />';
    sort($arr,SORT_NUMERIC);            //将数组的元素转换为数字进行排序
    print_r($arr);
    echo '<br />数组作为字符排序之后的信息：<br />';
    sort($arr,SORT_STRING);             //将数组的元素转换为字符进行排序
    print_r($arr);
?>
```

注意：在使用 sort 函数对字符串进行排序时候，会按照字符对应的 ASCII 值进行排序。对混合类型的数组进行排序的时候可能会产生无法预料的结果。

代码运行结果如图 5.16 所示。



图 5.16　运行结果

sort()和 rsort()函数的使用和理解都比较简单，这里就不再做详细讲解。

另一个与排序数组类似的函数是 shuffle()函数，它用来对数组进行随机排序，即将数组元素的顺序打乱，函数原型如下：

```
bool shuffle ( array &$array )
```

参数 array 即为需要进行随机排序的数组，它的使用非常简单。

【示例 5-17】以下代码演示使用 shuffle()函数对数组进行随机排序。

```
<?php
    $arr=array(3,23,'A','f','123','hello');    //定义一个数组
    echo '排序之前的数组信息：<br />';
    print_r($arr);
    shuffle($arr);                             //对数组进行随机排序
    echo '<br />排序之后的数组信息：<br />';
    print_r($arr);                             //输出数组信息
?>
```

代码运行结果如图 5.17 所示。

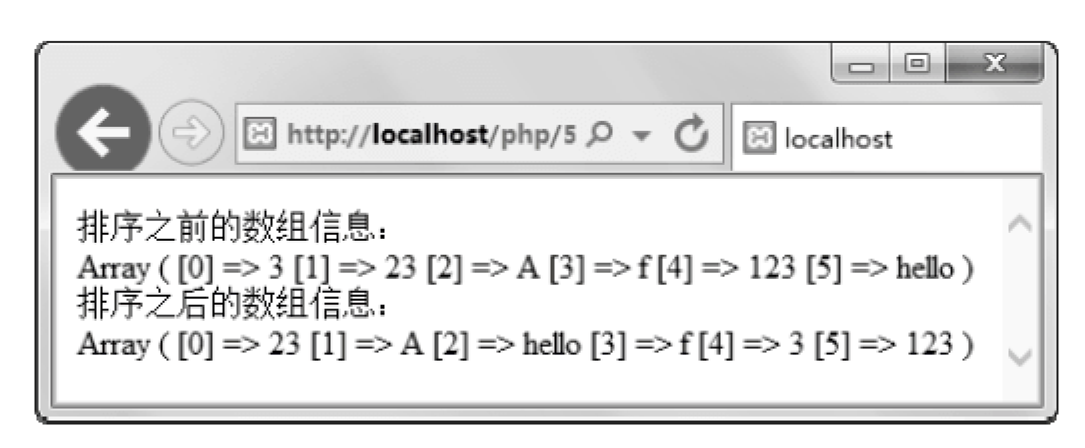


图 5.17　运行结果

由于这个函数的输出结果是随机的，因此每次的输出结果可能会不同。

5.4.2　过滤数组中的元素

过滤数组中的元素是一个非常有用的操作。例如，可以对分数进行过滤，将 100 以上的分数从数组中删除；还可以将混合类型数组中的字符串元素输出为一个新数组；也可以在数组中搜索指定的值等操作。

【示例 5-18】以下代码演示使用自定义函数删除数组中的偶数元素。

```
<?php
    function myfunc(&$arr){                        //自定义一个过滤函数
        $j=count($arr);
        for($i=0;$i<$j;$i++){
            if($arr[$i]%2==0)
                unset($arr[$i]);
        }
    }
    $arr=array(23,14,37,263,244,379,100,153,150);  //定义一个数组
    echo '数组进行过滤之前的信息：<br />';
    print_r($arr);
    myfunc($arr);                                  //调用自定义函数
    echo '<br />数组进行过滤之后的信息：<br />';
    print_r($arr);
?>
```

代码运行结果如图 5.18 所示。

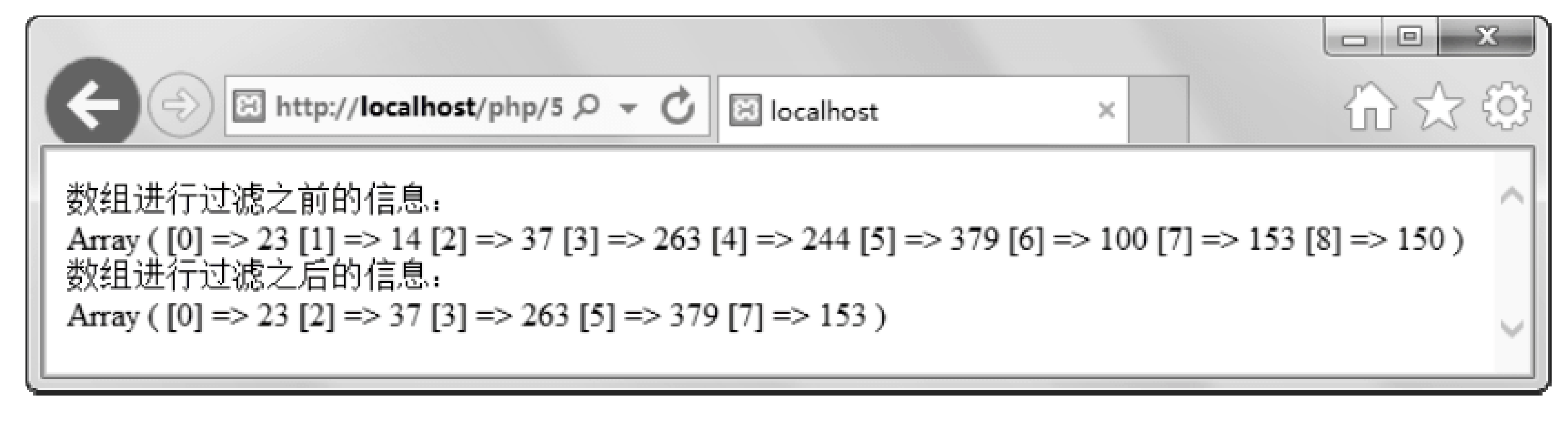


图 5.18　运行结果

从运行结果可以看出，我们的自定义函数成功过滤了数组中的偶数值。这个自定义的函数会修改原来的数组。下面我们来定义一个返回数组中 0～100 之间元素的函数，并且返回一个新的数组。

【示例 5-19】以下代码演示使用自定义函数，返回数组中 0～100 之间的元素组成的数组。

```
<?php
```

```
    function myfunc($arr){          //定义过滤函数
        $j=count($arr);
        for($i=0;$i<$j;$i++){
            if($arr[$i]>=0&&$arr[$i]<=100)
                $n_arr[]=$arr[$i];
        }
        return $n_arr;
    }
    $arr=array(2,34,5,175,168,94,942,105);
    echo '原来的数组信息：<br />';
    print_r($arr);
    $newarr=myfunc($arr);           //调用函数并使用变量接收函数的返回值
    echo '<br />过滤出来的新数组信息：<br />';
    print_r($newarr);               //输出新数组的信息
?>
```

代码运行结果如图 5.19 所示。

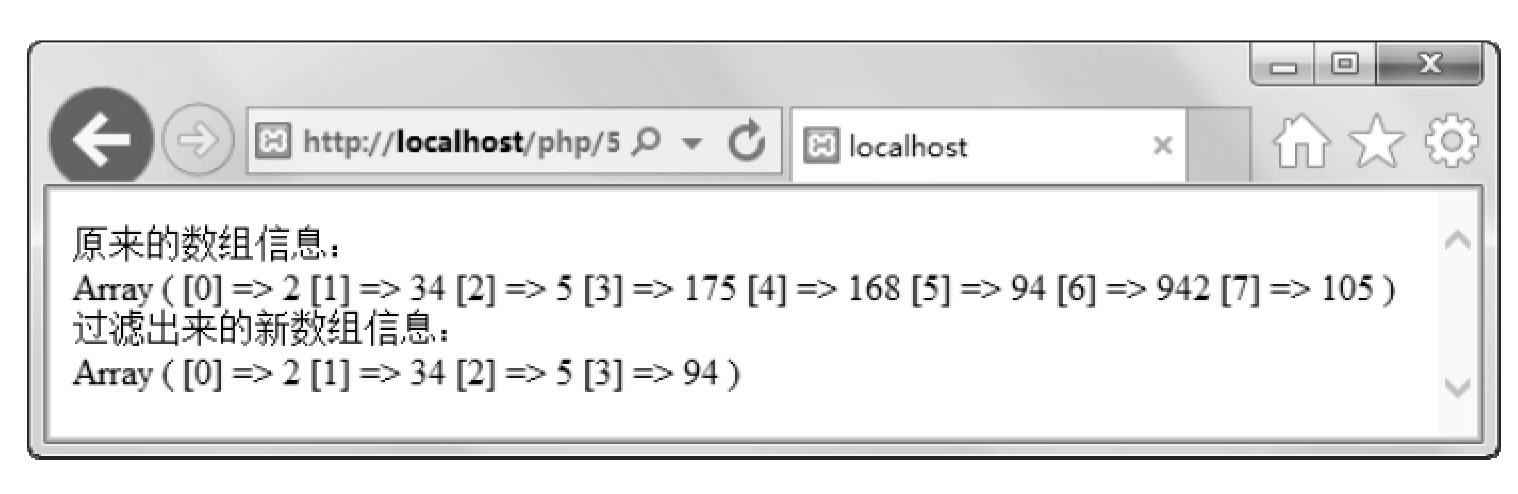


图 5.19　运行结果

除了我们可以这样自定义数组来过滤数组元素外，PHP 还提供了 array_filter()函数通过自定义的回调函数过滤数组中的元素，它的原型如下：

```
array array_filter ( array $input [, callback $callback ] )
```

参数 input 即为要进行过滤的数组；参数 callback 即为进行过滤的函数，如果回调函数返回 TRUE，则 input 数组的当前值会被包含在返回的结果数组中，数组的键名保留不变。

注意： *在使用 array_filter()函数过滤数组元素的时候不可增加或者删除数组的元素，否则会得到不可预料的结果。*

【示例 5-20】 以下代码演示使用 array_filter()函数过滤数组。

```
<?php
    function odd($x){                                    //定义过滤偶数的函数
        if($x%2==1)
            return TRUE;
    }
    function even($x){                                   //定义过滤奇数的函数
        if($x%2==0)
            return TRUE;
    }
    $arr=array(1,2,3,123,35,47,58,103,116);            //定义一个数组
    echo '过滤前的数组信息：<br />';
    print_r($arr);
    echo '<br />过滤掉偶数后的数组信息：<br />';
    print_r(array_filter($arr,'odd'));                  //调用函数对数组进行过滤
    echo '<br />过滤掉奇数后的数组信息：<br />';
    print_r(array_filter($arr,'even'));
```

```
?>
```

代码运行结果如图 5.20 所示。

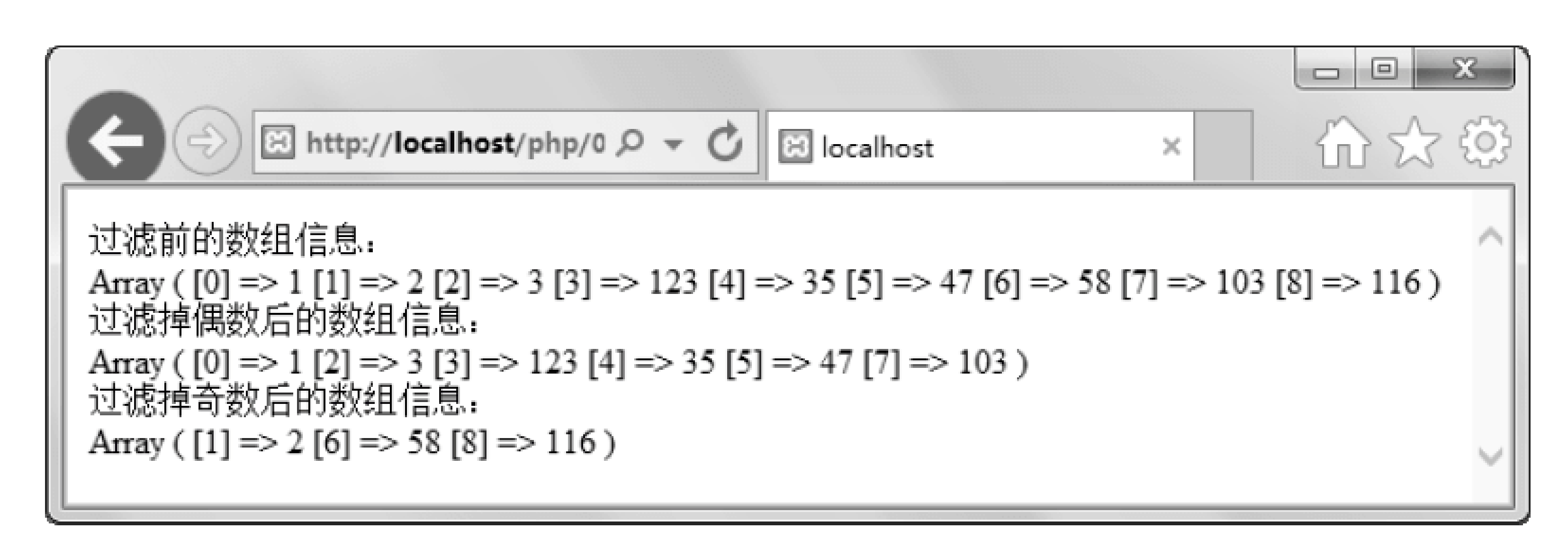


图 5.20　运行结果

当然，array_filter()函数不仅能用来做上面的事情，还可以使用该函数将数组中的整数过滤出来。

【示例 5-21】以下代码演示使用 array_filter()函数过滤数组中的非整数元素。

```
<?php
    function not_int($x){                          //定义过滤非整数的函数
        if(is_int($x))
            return TRUE;
    }
    $arr=array(1,2,'a',3,4,'c','d');               //定义数组
    echo '过滤前的数组信息：<br />';
    print_r($arr);                                 //输出过滤前的数组信息
    $res=array_filter($arr,'not_int');             //调用函数过滤数组
    echo '<br />过滤后的数组信息：<br />';
    print_r($res);                                 //输出过滤后的数组信息
?>
```

代码运行结果如图 5.21 所示。

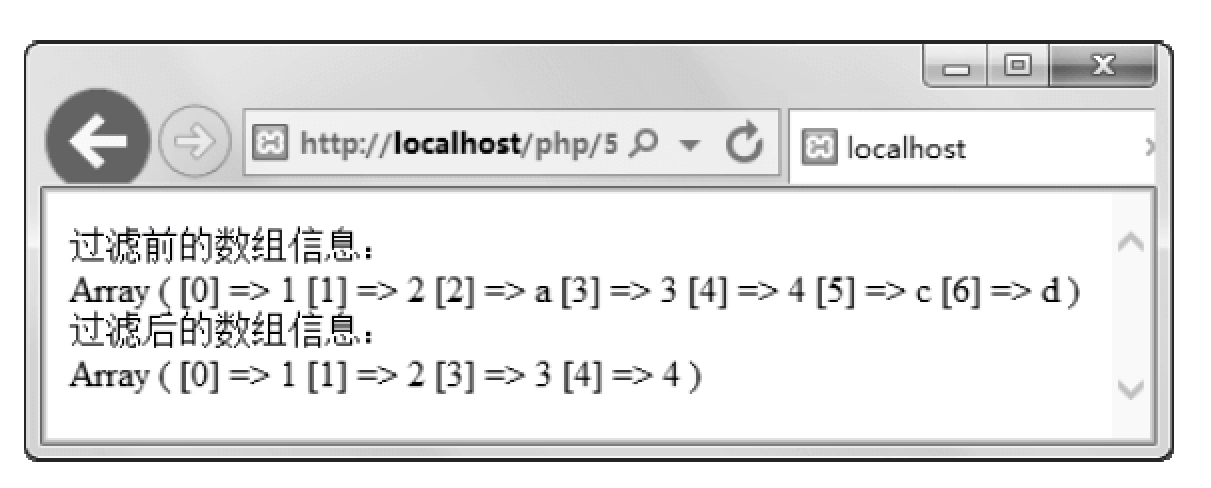


图 5.21　运行结果

从运行结果可以看到，使用 array_filter()函数成功过滤了数组中的非整数的元素。我们还可以过滤掉非字符的元素，这里就留给读者去实践。

5.5　关 联 数 组

我们知道，PHP 的数组索引也是可以自定义的，它允许的类型为整型和字符串形，这种使用自定义键名的数组就称为关联数组。关联数组的使用方法同索引数组多数是相同的，因此本节主要讲解定义关联数组、数组的比较运算符和遍历数组的其他一些方式。

5.5.1　定义关联数组

定义关联数组即为数组显式地指定索引名，指定的方式非常简单，通常的形式如下：

```
array(索引=>值…)
```

索引数组中同样可以有不显式指定索引的元素，被省略的索引按照索引数组的指定规则进行指定，我们可以通过一个示例来演示。

【示例 5-22】以下代码演示关联数组的定义方式和索引值的指定规则。

```
<?php
    $arr=array('name'=>'PHP','age'=>22,7=>25,33,21=>35);    //定义一个索引数组
    echo '数组$arr 的元素个数为：'.count($arr);    //输出索引数组的元素个数
    echo '<br />该数组的详细信息为：<br />';
    print_r($arr);                                  //输出数组的详细信息
?>
```

代码运行结果如图 5.22 所示。

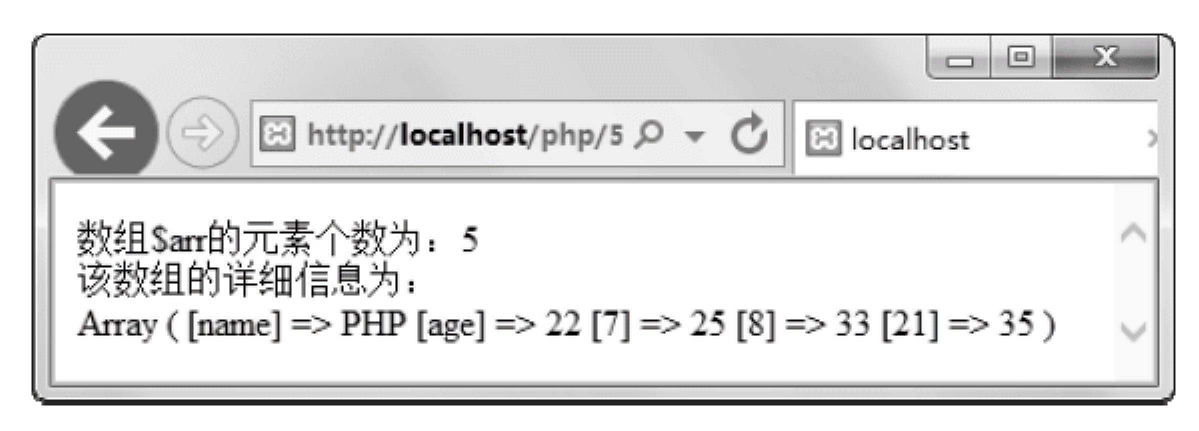


图 5.22　运行结果

在本示例中，需要读者注意的就是数组索引值的变化规律，还有就是数组元素的个数并不会因为数组指定了 21 作为索引而使得数组的长度变为 21。

5.5.2　数组比较运算符

PHP 中提供了专门用于对数组进行比较的运算符，如表 5.1 所示。

表 5.1　数组比较运算符

运算符	名　称	作　　用
==	相等	如果两个数组具有相同的索引/值则为 TRUE
===	全等	如果两个数组具有相同的索引/值对并且顺序和类型都相同则为 TRUE
!=	不等	如果两个数组不相等则为 TRUE
<>	不等	作用与!=相同
!==	不全等	如果两个数组不全等则为 TRUE

表 5.1 中的相等（==）和全等（===）运算符，这两个运算符的区别在于元素的顺序以及元素的类型。如下所示的两个数组：

```
$arra=array('hello','hi')
$arrb=array(1=>'hi',0=>'hello')
```

数组$arra 与数组$arrb 相等但不全等，不全等是因为它们的索引/值顺序不同。再来看

两个数组：

```
$arrc=array(1,2)
$arrd=array('1',2);
```

数组$arrc 与数组$arrd 同样相等但不全等，不全等是因为元素的类型不同。不等和不全等运算符的使用方法正好和它们相反，这里就不再赘述。

【示例 5-23】以下代码演示数组比较运算符的使用。

```
<?php
    $arra=array('hello','hi');           //定义多个数组
    $arrb=array(1=>'hi',0=>'hello');
    $arrc=array(1,2);
    $arrd=array('1',2);
    echo '$arra:';
    print_r($arra);                      //输出各个数组的信息以便读者对照
    echo '<br />$arrb:';
    print_r($arrb);
    echo '<br />$arrc:';
    print_r($arrc);
    echo '<br />$arrd:';
    print_r($arrd);
    echo '<br />$arra==$arrb:'.($arra==$arrb); //对数组进行比较运算并输出计算结果
    echo '<br />$arra===$arrb:'.($arra===$arrb);
    echo '<br />$arrc==$arrd:'.($arrc==$arrd);
    echo '<br />$arrc===$arrd:'.($arrc===$arrd);
    echo '<br />$arra!=$arrc:'.($arra!=$arrc);
    echo '<br />$arrb<>$arrd:'.($arrb<>$arrd);
?>
```

代码运行结果如图 5.23 所示。

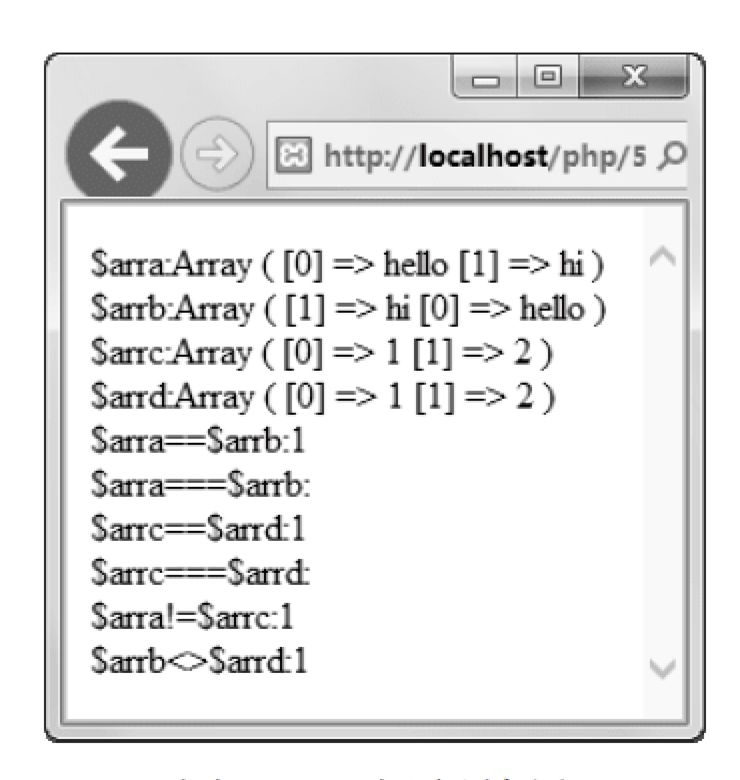


图 5.23　运行结果

读者可以从运行结果中对照比较各个数组，以理解这些数组运算符。

5.5.3　使用 foreach 结构遍历数组

前面我们已经使用 for 循环遍历过索引数组。我们知道关联数组的索引值通常是自定义的，因此通常不能通过 for 循环来遍历。foreach 是 PHP 中专门用来遍历数组的结构，它有两种使用形式，如下所示。

```
foreach (array_expression as $value)
```

```
    statement
foreach (array_expression as $key => $value)
    statement
```

第一种形式中的参数 array_expression 表示要进行遍历的数组；参数$value 用来接收当前元素的值。第二种形式是第一种形式的补充，除了使用参数$value 来接收当前元素的值外，还使用参数$key 来接收当前元素的键值。foreach 结构的工作原理是每次循环都会访问下一个数组元素。

数组指针是数组内部特有的指针，它初始会指向数组的第一个元素，可以通过数组指针操作函数来对指针进行操作。由于 foreach 控制的是数组的指针，因此可以遍历任意数组。

【示例 5-24】以下代码演示使用 foreach 循环遍历数组。

```
<?php
    $arr=array(63,'abc',45,'hello',3,7,9,'DEF');    //定义一个索引数组
    echo '遍历一个索引数组：<br />';
    foreach($arr as $v)                             //使用 foreach 遍历数组
        echo "{$v}  ";
    $arra=array('a'=>17,'B'=>'hello',123,'def','III'=>3);  //定义一个关联数组
    echo '<br />遍历一个关联数组：<br />';
    foreach($arra as $k=>$v)                        //使用 foreach 遍历数组
        echo "{$k}=>{$v}<br />";
?>
```

代码运行结果如图 5.24 所示。

从运行结果可以看到，使用 foreach 可以轻松完成遍历一个索引数组。当然 foreach 不止可以用来输出数组元素，下面就来演示使用 foreach 修改数组中元素的值。由于 foreach 默认是使用传值赋值，因此就可以以引用的形式来访问元素并修改其值。

【示例 5-25】以下代码演示使用 foreach 修改数组元素的值。

```
<?php
    $arr=array('I'=>1,'II'=>2,'III'=>3,'IV'=>4,'V'=>5);//定义一个数组
    echo '修改之前数组信息：<br />';
    print_r($arr);                                  //输出修改之前数组信息
    foreach($arr as &$v)
        $v=$v*2+1;                                  //使用遍历修改数组的值
    echo '<br />修改之后数组信息：<br />';
    print_r($arr);                                  //输出修改之后数组信息
?>
```

代码运行结果如图 5.25 所示。



图 5.24　运行结果

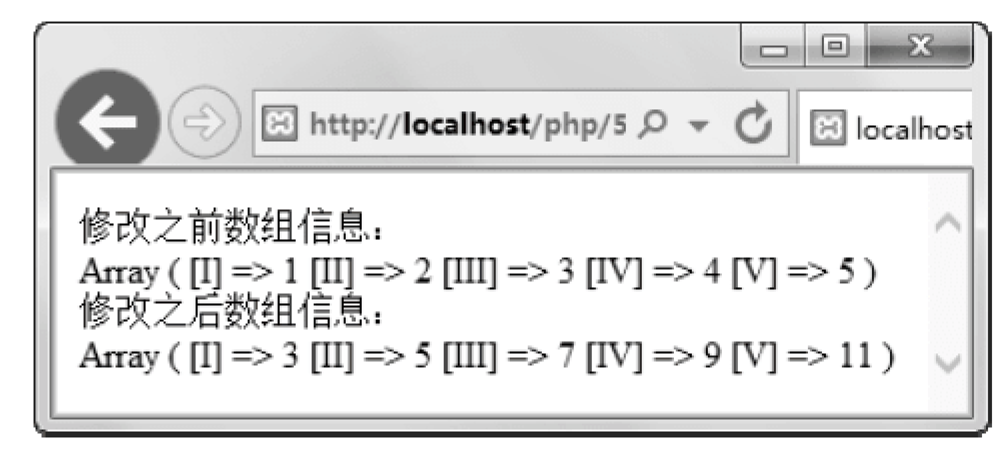


图 5.25　运行结果

从运行结果可以看到，使用 foreach 成功修改了原数组的元素值。

5.5.4　使用指针控制函数遍历数组

foreach 结构就是通过控制数组指针来遍历一个数组的。除此之外，PHP 中专门提供了多个指针控制函数，使用指针控制函数可以更加灵活地对数组进行操作。首先来介绍这些控制函数，然后再使用这些函数进行遍历数组及进行其他的操作。

1. each()和 list()函数

each()函数返回数组中当前的索引/值对并将数组指针向前移动一步，该函数的形式如下：

```
array each ( array &$array )
```

该函数会返回一个含有四个元素的数组，形式如下：

```
Array ( [1] =>值 [value] =>值 [0] => 索引 [key] => 索引)
```

该函数当数组指针移出数组时会返回 FALSE。

【示例 5-26】以下代码演示使用 each()函数输出数组元素的索引/值对。

```
<?php
    $arr=array(1,2,'III'=>3,'IV'=>4);          //定义一个数组
    for($i=0;$i<count($arr);$i++){
        print_r(each($arr));                   //循环输出数组的索引/值对
        echo '<br />';
    }
?>
```

代码运行结果如图 5.26 所示。

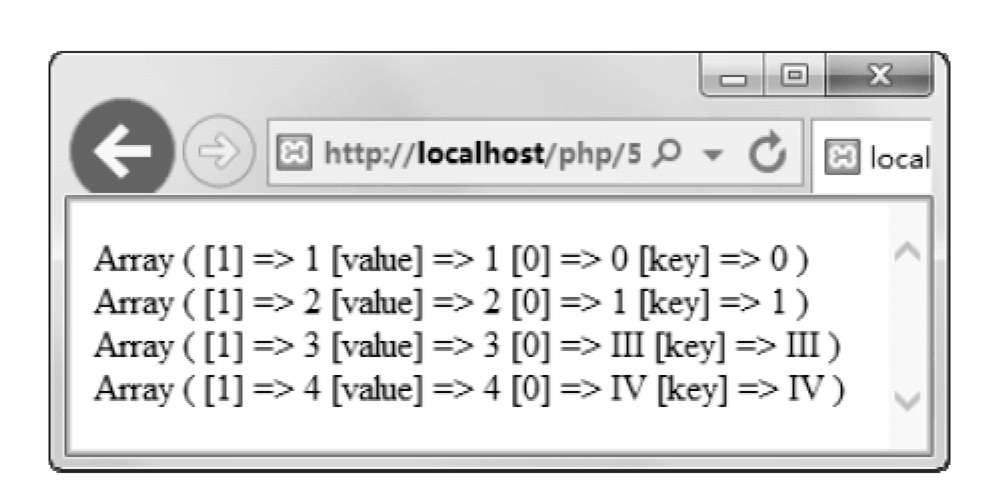


图 5.26　运行结果

以上代码之所以可以通过一个简单的循环来输出对应元素的索引/值对，是由于每次调用 each()函数均会向前移动指针。

List()函数用来把数组中的值赋给一些变量。

```
array list ( mixed $varname [, mixed $... ] )
```

函数中的参数 varname 用来接收数组对应的元素，而且只能接收键为数字的元素，即 list 的第一个参数会接收数组中最小键值对应的值。

【示例 5-27】以下代码演示使用 list()函数接收 each()函数返回的数组。

```
<?php
    $arr=array(1,2,'III'=>3,'IV'=>4);          //定义一个数组
    list($k,$v)=each($arr);           //使用 list()函数接受 each()函数返回的值
```

```
    echo "{$k}=>{$v}";          //输出变量 k 和 v 的值
?>
```

代码运行结果如图 5.27 所示。

运行结果中输出的是数组$arr 的第一个元素的索引/值对，因此就可以通过循环来依次输出数组的键值对。

【示例 5-28】以下代码演示使用 while 循环联合 list()和 each()函数遍历数组。

```
<?php
    $arr=array(1,2,'III'=>3,'IV'=>4);   //定义一个数组
    while(list($k,$v)=each($arr)){      //使用 each()函数返回 FALSE 的特性
                                          作为判断条件
        echo "{$k}=>{$v}<br />";        //输出数组的索引/值对
    }
?>
```

代码运行结果如图 5.28 所示。

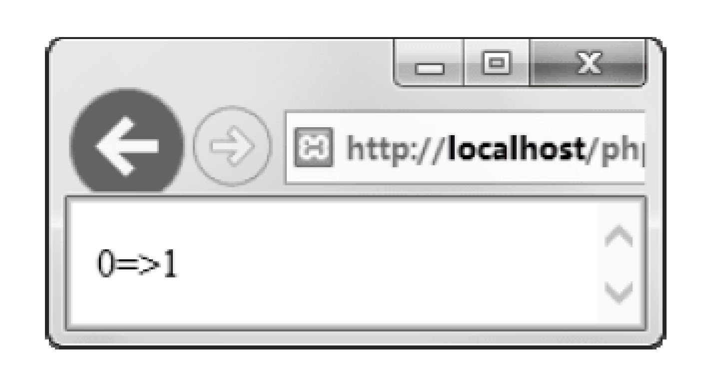


图 5.27　运行结果

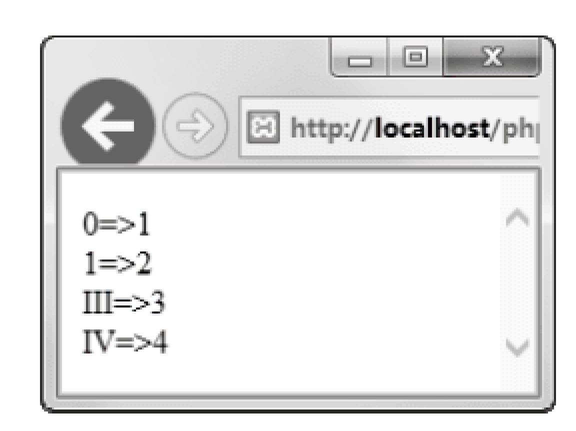


图 5.28　运行结果

从运行结果可以看到成功遍历输出了数组的索引/值对，当然也可以使用 for 循环来遍历，这里就不再详细讲解，读者可以自己尝试。

2. 数组指针控制函数

PHP 中提供了多个指针控制函数，下面就分别介绍这些函数。

- current()：返回当前数组指针指向的元素，如果数组指针指向超出了单元列表的末端，则返回 FALSE，它的原型如下：

```
mixed current ( array &$array )
```

- end()：将数组的内部指针指向最后一个元素并返回其值，它的原型如下：

```
mixed end ( array &$array )
```

- next()：将数组中的内部指针向前移动一位并返回其值，当没有更多单元时返回 FALSE，它的原型如下：

```
mixed next ( array &$array )
```

- prev()：将数组的内部指针倒回一位并返回其值，当没有更多单元时返回 FALSE，它的原型如下：

```
mixed prev ( array &$array )
```

- reset()：将数组的内部指针指向第一个元素并返回其值，如果数组为空则返回 FALSE，它的原型如下：

```
mixed reset ( array &$array )
```

这些数组指针控制函数的加入，使得 PHP 的数组使用可以极为灵活。我们可以通过使用指针控制函数很方便地访问任何数字的任何元素。

【示例 5-29】以下代码演示使用自定义函数访问数组的元素。

```
<?php
    function return_item($arr,$num=0){       //定义函数
        for($i=0;$i<$num;$i++){              //循环向前移动数组指针
            next($arr);
        }
        echo "<br />第{$num}个元素为: ".current($arr);  //输出当前数组指针
                                                          指向的元素
    }
    $arr=array('I'=>'hello','II'=>2,'III'=>'D','IV'=>376,358);
                                             //定义一个数组
    echo '输出数组的详细信息: ';
    return_item($arr);                       //调用函数并传入参数输出对应的数组元素
    return_item($arr,1);
    return_item($arr,2);
    return_item($arr,3);
    return_item($arr,5);    //由于数组元素只有 5 个，因此输出第 6 个元素为空
?>
```

代码运行结果如图 5.29 所示。

上面的代码演示了使用 next()函数向前移动数组指针来输出数组元素的值，还可以通过扩展自定义函数来使用 prev 函数向后移动指针来返回数组元素的值。

【示例 5-30】以下代码演示通过扩展示例 5-29 中的自定义函数，获得更多的功能。

```
<?php
    function return_item($arr,$num=0){       //定义函数
        if($num<0){
            end($arr);                       //将数组指针指向最后一个元素
            for($i=0;$i<abs($num)-1;$i++){   //abs 函数用于取得变量的绝对值
                prev($arr);
            }
            echo "<br />第{$num}个元素为: ".current($arr);
                                             //输出当前数组指针指向的元素
        }else{
            for($i=0;$i<$num;$i++){          //循环向前移动数组指针
                next($arr);
            }
            echo "<br />第{$num}个元素为: ".current($arr);
                                             //输出当前数组指针指向的元素
        }
    }
    $arr=array('I'=>'hello','II'=>2,'III'=>'D','IV'=>376,358);
                                             //定义一个数组
    echo '输出数组的详细信息: ';
    return_item($arr);                       //调用函数并传入参数输出对应的数组元素
    return_item($arr,-1);
    return_item($arr,2);
    return_item($arr,1);
    return_item($arr,-2);
?>
```

代码运行结果如图 5.30 所示。



图 5.29　运行结果



图 5.30　运行结果

以上的代码中的自定义函数通过接受整数和负数参数，实现了从数组的正向和反向来输出数组元素。这里只简单展现了数组控制指针的灵活之处，读者可以通过自己扩展函数来获得更加灵活的功能。

5.6　多 维 数 组

我们知道 PHP 中的数组可以存储多种类型的数据，其中就包括数组。本节将要讲解的是数组作为数组元素的形式，这种形式的数组被称为多维数组。常用的多维数组是二维数组和三维数组，我们将主要介绍二维数组。

5.6.1　二维数组的优势

我们知道，一个学生有多个科目的成绩，那么就可以用如下形式来保存一个班里多个学生的成绩：

```
$stu01=array(76,87,68,…,98);
$stu02=array(65,89,95,…,76);
$stu03=array(90,80,66,…,60);
…
$stun=array(90,95,65,…,100);
```

我们可以想到，如果一个班级有 50 个学生的话，就需要声明 50 个变量来存储他们的成绩，这就类似于本章开头的例子。那么自然就可以想到我们同样可以使用数组来保存这些数据，只不过这个数组的元素也是一些数组而已，我们可以改写为如下的形式：

```
$stu=array(array(76,87,68,…,98),
        array(65,89,95,…,76),
        array(90,80,66,…,60),
        …
        array(90,95,65,…,100));
```

从上面的形式可以看到，省去了多个变量的声明，而且访问其中的成绩也与一维数组无异。

5.6.2　访问二维数组的元素

访问二维数组的元素同样使用如下的形式：

```
数组名[索引]
```

使用上面的形式访问到的通常会是一个数组。

【示例 5-31】以下代码演示访问二维数组的元素。

```
<?php
    $stu=array(array(76,87,68),
                array(65,89,95),
                array(90,80,66),
                array(90,95,65));        //定义一个二维数组
    echo '输出$stu 数组的第一个元素：<br />';
    print_r($stu[0]);   //由于访问到的将会是一个数组，因此使用 print_r 来输出
    echo '<br />输出$stu 数组的第二个元素：<br />';
    print_r($stu[1]);
    echo '<br />输出$stu 数组的第三个元素：<br />';
    print_r($stu[2]);
    echo '<br />输出$stu 数组的第四个元素：<br />';
    print_r($stu[3]);
?>
```

代码运行结果如图 5.31 所示。

从运行结果可以看到，代码成功访问到了数组中的元素，并输出其信息。

由于使用“数组名[索引]”形式访问到的是一个数组，因此可以将其作为数组名来访问其中的元素，形式如下：

```
数组名[索引][索引]
```

【示例 5-32】以下代码演示访问二维数组中数组的元素。

```
<?php
    $stu=array(array(76,87,68),
                array(65,89,95),
                array(90,80,66),
                array(90,95,65));        //定义一个二维数组
    echo '$stu 数组的第 1 个数组元素的第一个元素为：'.$stu[0][0];
                                        //访问数组中的指定元素
    echo '<br />$stu 数组的第 2 个数组元素的第二个元素为：'.$stu[1][1];
    echo '<br />$stu 数组的第 3 个数组元素的第三个元素为：'.$stu[2][2];
?>
```

代码运行结果如图 5.32 所示。



图 5.31　运行结果



图 5.32　运行结果

从运行结果中我们可以看到，使用“数组名[索引][索引]”的形式正确访问到了指定的数组元素，这也是二维数组的重点知识。二维数组同样可以为关联数组即可以自定义索引名，这里就不做讲解。

5.6.3　遍历二维数组

二维数组通常使用 foreach 来遍历，对于规则的二维数组，也就是说数组的元素均为数组的二维数组，通常的形式如下：

```
array(array(…),array(…),array(…),array…)
```

这种形式的二维数组我们可以通过使用 foreach 嵌套来实现遍历数组元素。

【示例 5-33】以下代码演示使用 foreach 嵌套来遍历二维数组。

```
<?php
    $stu=array(array(76,87,68),
                array(65,89,95),
                array(90,80,66),
                array(90,95,65));       //定义一个二维数组
    foreach($stu as $k=>$v){            //遍历数组
        echo "元素{$k}中的元素：<br />";
        foreach($v as $k=>$v){
            echo "  {$k}=>{$v}";         //输出数组的索引/值对
        }
        echo '<br />';
    }
?>
```

代码运行结果如图 5.33 所示。

从运行结果可以看出，我们遍历输出了数组的所有元素。但是这段代码不能用来遍历一个不规则的二维数组即数组中有的元素不为数组的二维数组，通常的形式如下：

```
array(array(…),$value…,array(…)…)
```

如果要遍历这种二维数组，可以在 foreach 中添加判断来遍历，也可以通过递归来遍历，这里我们只介绍使用递归的方式。

【示例 5-34】以下代码演示使用递归遍历二维数组。

```
<?php
    $arr=array(array(76,87,68),
                array(65,89,95),
                array(90,80,66),
                array(90,95,65),5,234,56,'Hello'); //定义一个二维数组
    function ergodic($arr){                        //定义遍历数组的函数
        foreach($arr as $k=>$v){
            if(is_array($v)){
                echo "<br />数组元素{$k}中的元素：<br />";
                ergodic($v);                       //递归调用
                echo '<br />';
            }else{
                echo "{$k}=>{$v}<br />";
            }
        }
    }
```

```
        ergodic($arr);                                    //调用函数遍历数组
?>
```

代码运行结果如图 5.34 所示。



图 5.33　运行结果

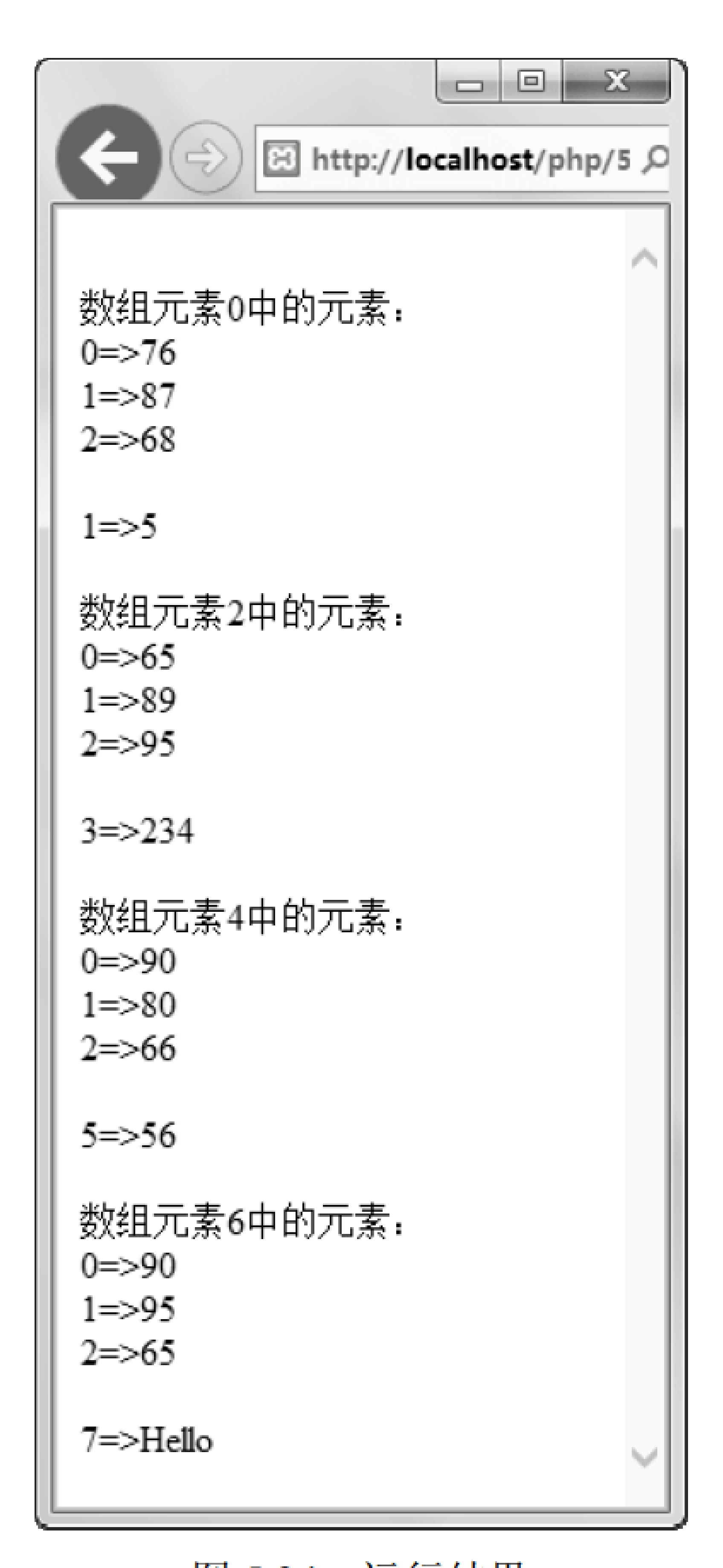


图 5.34　运行结果

从运行结果可以看到，使用我们的自定义函数成功遍历了不规则的二维数组。其他遍历的方法这里就不再详讲，读者可以根据自己情况来自己编写。

5.6.4　三维数组

有了前面学习二维数组的基础，大部分读者一定能够推导出三维数组。三维数组即将多个二维数组作为元素的数组，通常的形式如下：

```
array(array(array(…)…)…)
```

三维数组可以存放更多的信息，例如一个学生有多个成绩，多个学生可以合成为一个班，多个班级可以合成为一个年级，那么我们就可以创建一个如下形式的三维数组，保存这些信息：

```
$grade=array('class1'=>array('stu1'=>array('yuwen'=>85,'shuxue'=>95,
                 'yingyu'=>96),
                 'stu2'=>array('yuwen'=>76,'shuxue'=>89,'yingyu'=>99),
                 'stu3'=>array('yuwen'=>75,'shuxue'=>99,'yingyu'=>100),
                ...),
```

```
               'class2'=>array('stu1'=>array('yuwen'=>99,'shuxue'=>100,
                                 'yingyu'=>100),
                                 'stu2'=>array(...),
                                  ...),
               'class3'=>array(...),
               ...
               )
```

三维数组元素的访问同二维数组类似，通过如下形式即可访问：

```
数组名[索引][索引][索引]
```

【示例 5-35】以下代码演示使用访问三维数组中的元素。

```
<?php
    //定义一个三维数组
    $grade=array('class1'=>array('stu1'=>array('yuwen'=>85,
                                  'shuxue'=>95,'yingyu'=>96),
                                  'stu2'=>array('yuwen'=>76,
                                  'shuxue'=>89,'yingyu'=>99),
                                  'stu3'=>array('yuwen'=>75,
                                  'shuxue'=>99,'yingyu'=>100)),
                 'class2'=>array('stu1'=>array('yuwen'=>99,
                                  'shuxue'=>100,'yingyu'=>100)),
                 'class3'=>array(array(80,90,99)));
    //访问数组中的元素
    echo '该年级 1 班的 stu1 学生的 yuwen 成绩为:
                            '.$grade['class1']['stu1']['yuwen'];
    echo '<br />该年级 1 班的 stu3 学生的 yingyu 成绩为:
                            '.$grade['class1']['stu3']['yingyu'];
    echo '<br />该年级 2 班的 stu1 学生的 shuxue 成绩为:
                            '.$grade['class2']['stu1']['shuxue'];
    echo '<br />该年级 3 班的学生 0 的成绩 1 为: '.$grade['class3'][0][1];
?>
```

代码运行结果如图 5.35 所示。

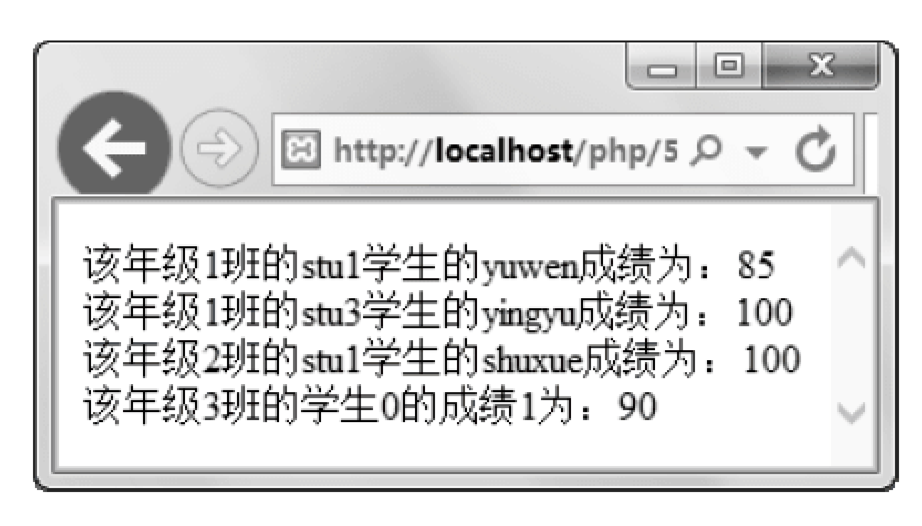


图 5.35　运行结果

三维数组的其他操作方法都同一维数组和二维数组类似，这里就不再详细讲解。

5.7　系统预定义数组

系统预定义数组同系统预定义常量类似，即这些数组是由系统定义的，我们可以直接来访问而无须定义，PHP 中常用的预定义数组如表 5.2 所示。

这些数组的使用方法同普通数组的使用方法相同，这里就不再详细讲解。

表 5.2　常用预定义数组

预定义数组名	说　　明
$GLOBALS	包含了全部变量的全局组合数组。变量的名字就是数组的键
$_SERVER	包含了诸如头信息、路径、以及脚本位置等信息的数组
$_GET	通过 URL 参数传递给当前脚本的变量的数组
$_POST	通过 HTTP POST 方法传递给当前脚本的变量的数组
$_REQUEST	默认情况下包含了$_GET，$_POST 和$_COOKIE 的数组
$_FILES	通过 HTTP POST 方式上传到当前脚本的项目的数组
$_SESSION	当前脚本可用 SESSION 变量的数组
$_COOKIE	通过 HTTP Cookies 方式传递给当前脚本的变量的数组
$_ENV	通过环境方式传递给当前脚本的变量的数组

5.8　小　　结

本章主要进行了数组部分的学习。PHP 的数组是非常强大的，因此本章尽可能详细地讲解到每一个知识点。数组中的索引数组是最常用的形式，因此花费了大篇的章节来讲解，这也是读者要掌握的主要部分。关联数组虽然可以使得数组看起来更加直观，但是在实际的使用过程中使用还是比较少的。本章的重点就在索引数组以及相关数组的操作，读者应该熟练掌握。

5.9　本 章 习 题

1. 将表 5.3 中的数据存储在一个名为$name 的索引数组中。

表 5.3　将以下数据存储在数组中

Tom	Jerry	Anna	Jim

2. 将表 5.4 中的数据存储在一个名为$email 的关联数组中。

表 5.4　将以下数据存储在数组中

Name	Tom	Jerry	Anna	Jim
Email	tom@gmail.com	jerry@gmail.com	Anna@yahoo.com	jim@hotmail.com

3. 将$email 数组中 Tom 的邮箱地址改为 tom@hotmail.com。
4. 将数组$name 中的元素颠倒位置传入一个新数组$newname 中。

第 6 章　面向对象编程

自 PHP 5 开始，PHP 就提供了完善的面向对象支持。面向对象思想可以使得程序更加符合人类看待事物的规律。同时，面向对象的引入使得程序的代码更简洁、更易于维护，并且具有更强的可重用性。面向对象编程的难点不在于语法，而在于如何将一个事物正确地抽象成一个类。

6.1　类 与 对 象

类可以理解为一些属性和能力的集合。例如，人是一个类，即人类。猫也是一个类，即猫类。PHP 中的类也是类似的，它是一个属性和方法的集合。对象即这个类的一个实例。例如每一个人都是人类的一个实例，每一只猫都是猫类的一个实例，而这些实例都是相互独立的，一个实例的变化不会影响到其他的实例。这就是类与对象关系的一个形象的说明。

6.1.1　抽象出一个类

在本章的开头我们就说过，面向对象的难点就在于如何正确地抽象出一个类。本节我们就以演示多个事物来抽象出一个类。

1. 人类

首先是我们最熟悉的人类，我们首先需要为这个类定义一个名称以此来称呼这个类，当然这个名称没有太多的限制，读者可以随意指定，这里我们指定为人类。我们首先可以总结出人的一些共有的特性，这里我们只随意指定几个：

- 有高智商；
- 可以直立行走；
- 擅长使用工具；
- 身高；
- 体重；
- 性别；
- 生日。

有了上面这些属性，就可以抽象出一个类了。现在回到 PHP 代码，PHP 中定义一个类使用的关键字是 class，定义一个类的语法结构如下：

```
class 类名{
    类的属性和能力;
}
```

我们可以看到类的定义方法还是非常简单的，那么就可以以伪代码的形式来定义人类，如下所示。

```
class 人类{
    有高智商;
    可以直立行走;
    身高;
    体重;
    性别;
    生日;
}
```

就像上面这样，我们就以代码的形式定义了一个人类，当然这段代码在 PHP 中是不可以使用的，但是只需要进行很少的更改即可成为正确的 PHP 代码。

2. 猫类

定义一个类最重要的就是找出这个类的实例所共有的特征。例如有一只猫天生就没有尾巴，而我们却不可以将这个特性抽象到猫类中，因为极大多数的猫都是有尾巴的。当然我们完全可以抽象一个类，这个类的特点就是没有尾巴。下面我们就来找出猫的一些特性：

- 四条腿；
- 有尾巴；
- 昼伏夜出；
- 会捉老鼠；
- 较强的攀爬能力。

同样，可以就这些特性定义一个类，如下所示。

```
class 猫类{
    四条腿;
    有尾巴;
    昼伏夜出;
    会捉老鼠;
    较强的攀爬能力;
}
```

类的名字并没有太多的约束，因此我们完全可以将具有以上特点的类定义为牛类，如下所示。

```
class 牛类{
    四条腿;
    有尾巴;
    昼伏夜出;
    会捉老鼠;
    较强的攀爬能力;
}
```

如上面的定义的牛类是完全正确的。

3. 有尾巴的动物类

类的定义并非是只可将一类实例的特点定义为一个类，我们完全可以定义一个例如有

尾巴的动物类。我们将这个类定义如下的特点：

- 有尾巴；
- 有四条腿；
- 是动物。

它的定义可以如下的伪代码表示：

```
class 有尾巴的动物{
    有尾巴;
    有四条腿;
    是动物;
}
```

以上我们就定义了一个有尾巴的动物类，它的实例可以是猫，牛，羊，长颈鹿和狮子等动物。而蛇不可以属于这个类，因为它没有四条腿；毛绒玩具狗也不属于这个类，因为它不是动物。

以上就是抽象出的多个类的过程，我们可以看到理解起来还是非常的简单的，这也就印证了面向对象会使得代码更加符合人类看待事物的规律的特性。

6.1.2　实例化一个类

类是一些属性和方法的集合，我们不可以直接拿来使用。这个理解起来也是比较容易的，就像人类，它是一个概念，是一个规定。它规定了人类必须有哪些属性和能力才能算是人类，而它并不是一个实例。符合一个类的实例可以被称为类的对象；而我们也可以根据类来实例化一个对象。在 PHP 中更多的是根据类来实例化一个对象。实例化类的对象使用的关键字是 new，实例化对象的语法如下：

```
new 类名(参数)
```

实例化一个猫类的对象，代码如下：

```
new 猫类();
```

同样可以实现有尾巴的动物类的多个实例，代码如下：

```
new 有尾巴的动物类();
new 有尾巴的动物类();
new 有尾巴的动物类();
```

以上的 3 行代码就实例化了有尾巴的动物类的 3 个对象。以上都是使用伪代码的形式来演示的，下面我们就以实际的 PHP 代码来演示这一过程。由于现在我们还没有学习类中定义属性和方法的知识，因此这里我们就只以一个空类来讲解。空类即没有任何属性和方法的类，如下所示。

```
class 类名{

}
```

但是在 PHP 中这是一个合法的类，我们可以将它实例化。

【示例 6-1】以下代码演示实例化一个类的对象。

```
<?php
    class ren{                  //定义人类

    }
    class mao{                  //定义猫类

    }
    new ren();                  //实例化人类
    new mao();                  //实例化猫类
    new mao();                  //实例化猫类
    new mao();                  //实例化猫类
?>
```

运行上代码没有任何输出，那么我们如何才能知道确实实例化了这些类呢？我们可以使用 var_dump 函数输出这些实例化代码的类型来验证。

【示例 6-2】以下代码演示使用 var_dump()函数验证类是否被实例化。

```
<?php
    class ren{                  //定义人类

    }
    class mao{                  //定义猫类

    }
    echo '实例化一个人类：';
    var_dump(new ren());        //实例化人类
    echo '<br />实例化两个猫类：';
    var_dump(new mao());        //实例化猫类
    var_dump(new mao());        //实例化猫类
?>
```

以上代码的运行结果如图 6.1 所示。

从运行结果的输出类型中的 object(ren)即表示为 ren 类的一个对象，而 object(mao)则表示 mao 类的一个对象。

从示例 6-2 我们可以看出这些类确实被实例化成了对象，但是很多的读者就会就此疑惑了，因为实例化对象所用的代码完全一样，怎么才能分别使用这些对象呢？其实这些对象是有区别的，只不过是这些区别只有 PHP 核心知道。这就类似于手机生产商生产手机，通常在出厂之前会有一个内部的工程机代号。使用 new 关键字实例化一个类后会返回一个独一无二的标识，那么我们就可以使用一个变量来保存这个代号，就可以分别标识每个对象了，代码形式如下：

```
变量=new 类名(参数列表)
```

【示例 6-3】以下代码演示使用变量来标识一个对象。

```
<?php
    class ren{              //定义人类

    }
    class mao{              //定义猫类

    }
    $ren1=new ren();        //实例化人类
```

```
        $mao1=new mao();           //实例化猫类
        $mao2=new mao();           //实例化猫类
        echo '输出变量$ren1 的类型：';
        var_dump($ren1);           //查看变量的类型
        echo '<br />输出变量$mao1 的类型：';
        var dump($mao1);
        echo '<br />输出变量$mao1 的类型：';
        var_dump($mao1);
    ?>
```

代码运行结果如图 6.2 所示。



图 6.1　运行结果



图 6.2　运行结果

从以上代码的运行结果可以看出，这些变量正确保存了对象的标识符。

6.1.3　类的成员

在前面小节中将事物抽象成一个类的时候会总结出一些实例的共有特性，这些特性就组成了类的成员。例如猫类的：

- ❑ 四条腿；
- ❑ 有尾巴；
- ❑ 昼伏夜出；
- ❑ 会捉老鼠；
- ❑ 较强的攀爬能力。

这些特性就是猫类的成员。而根据成员的特点，我们又将成员分为成员属性和成员方法。例如，描述属性的成员我们就可以将其归为成员属性，表现在 PHP 代码中就是变量或者常量。例如攀爬、捉老鼠这些能力我们就将其归为成员方法，表现在 PHP 代码中就是函数。

6.2　成 员 属 性

成员属性通常在类中表示该类的对象应有的属性，它通常为变量或者常量。下面我们来详细学习类的成员属性。

6.2.1　变量属性

为了学习本小节的内容，我们首先来简单定义一个猫类，它有如下的成员：

- ❑ 年龄（age）；
- ❑ 体重（weight）；
- ❑ 毛色（color）。

将以上这些成员构成的类如下所示。

```
class mao{
    age;
    weight;
    color;
}
```

但是这并不是正确的 PHP 代码。我们在将其改写为正确的 PHP 代码前首先需要学习一些新的知识。

1. 访问控制标识符

PHP 中规定，成员属性必须在前面添加访问控制标识符来定义该属性的可访问性。PHP 中的访问控制符，如表 6.1 所示。

表 6.1　PHP 中的访问控制标识符

关键字	说　　明
public	由 public 所定义的类成员可以在任何地方被访问
protected	由 protected 所定义的类成员可以被其所在类以及子类和父类访问
private	由 private 定义的类成员则只能被其所在类访问

在认识了访问控制关键字以后，我就可以将前面定义的猫类更正为正确的 PHP 代码了，如下所示。

```
class mao{
    public $age;
    protected $weight;
    private $color;
}
```

以上代码就是一段正确的 PHP 代码，下面我们就来实例化一个该类的对象并且使用 var_dump()函数输出该对象的成员。

【示例 6-4】以下代码演示实例化一个 mao 类的对象，并使用 var_dump 函数输出该对象的成员。

```
<?php
    class mao{                  //定义猫类
        public $age;            //定义多个成员属性
        protected $weight;
        private $color;
    }
    $mao1=new mao();            //实例化一个对象
    echo '输出该对象的成员：<br />';
    var_dump($mao1);            //输出对象的成员
?>
```

代码运行结果如图 6.3 所示。

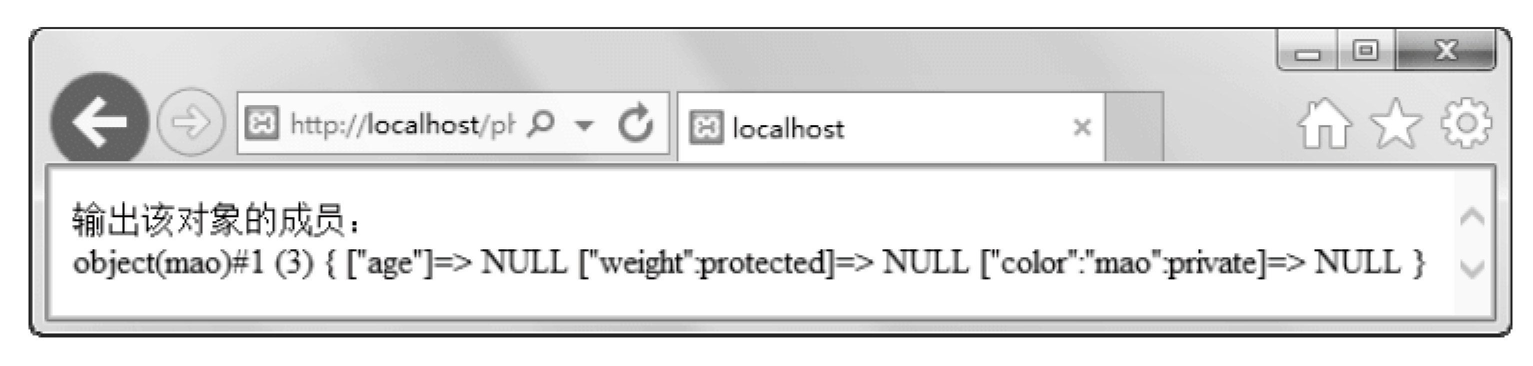


图 6.3　运行结果

从以上代码的输出结果可以看到该对象的所有属性以及该属性的可访问性。

2. 访问对象成员

访问对象的成员需要使用“->”符号来访问，它的用法如下：

```
对象->对象的成员
```

下面就通过一个示例来演示访问对象成员。

【示例 6-5】以下代码演示访问对象成员及访问控制标识符对成员的影响。

```
<?php
    class mao{                //定义猫类
        public $age;          //定义多个成员属性
        protected $weight;
        private $color;
    }
    $mao1=new mao();          //实例化一个对象
    echo '输出对象$mao1 的 age 属性：'.$mao1->age;
    $mao1->age=3;             //给对象的 age 属性赋值
    echo '<br />再次输出$mao1 的 age 属性：'.$mao1->age;
?>
```

注意：在使用“->”运算符访问类成员属性的时候，成员属性名前不可有“$”符。

代码运行结果如图 6.4 所示。

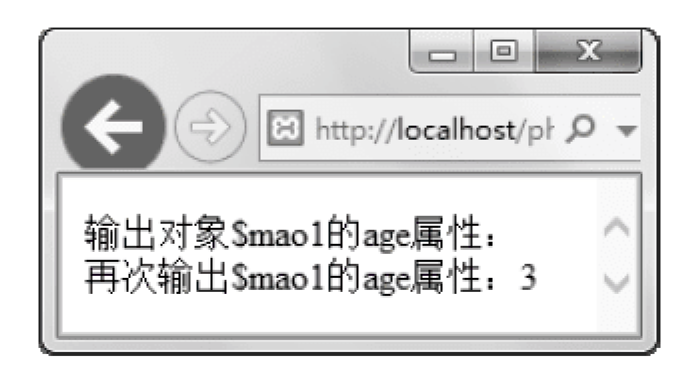


图 6.4　运行结果

访问控制标识符的作用就是控制成员的可访问性，下面我们来演示访问控制运算符对对象成员的影响。

【示例 6-6】以下代码演示访问控制运算符对对象成员的影响。

```
01  <?php
02      class mao{                                       //定义猫类
03          public $age;                                 //定义多个成员属性
04          protected $weight;
05          private $color;
06      }
07      $mao1=new mao();                                 //实例化一个对象
08      echo '输出对象 mao1 的 age 成员：'.$mao1->age;   //访问对象的成员
09      echo '<br />尝试输出对象 mao1 的 weight 成员：'.$mao1->weight;
10      echo '<br />尝试输出对象 mao1 的 color 成员：'.$mao1->color;
11  ?>
```

代码运行结果如图 6.5 所示。

我们可以看到，当程序运行到代码的第 9 行的时候报出了一个致命的错误，错误原因就是“不能访问保护成员”。protected 成员及 private 成员的访问我们将在后面的小节中讲解。



图 6.5　运行结果

3. 初始化类的成员属性

在之前定义的类中，我们定义的类的成员属性均为空。这些成员属性在定义的时候是可以初始化为一个常量的，这就如同为对象的属性规定了一个默认值一样。下面就来重新定义“mao”类，如下所示。

```
class mao{
    public $age=0;
    public $weight=50;
    public $color=' white';
}
```

🔔注意：为了方便讲解，我们将该类所有的成员访问标识符均定义为 public。

这样为属性初始化一个值导致的结果就是，该类所有的对象的属性都相同，就类似于所有出生的小猫年龄都为 0、体重都为 50 克、毛色都为白色一样。

【示例 6-7】以下代码演示实例化有初始值的类的多个对象，并访问它们的属性。

```
<?php
    class mao{                  //定义猫类
        public $age=0;          //定义多个属性并初始化
        public $weight=50;
        public $color='white';
    }
    $mao1=new mao();            //实例化猫类的多个对象
    $mao2=new mao();
    //输出 mao1、mao2 对象的属性
    echo "mao1 的年龄为{$mao1->age}，体重为{$mao1->weight}，毛色为
    {$mao1->color}。<br />";
    echo "mao2 的年龄为{$mao2->age}，体重为{$mao2->weight}，毛色为
    {$mao2->color}。<br />";
?>
```

代码运行结果如图 6.6 所示。

从以上代码的输出结果可以看出，该类实例化的对象都有完全相同的属性，当然这些属性都是可以修改的，下面就来演示。

【示例 6-8】以下代码演示实例化有初始值的类的多个对象，并修改其中一个对象的属性值。

```
<?php
    class mao{                  //定义猫类
        public $age=0;          //定义多个属性并初始化
        public $weight=50;
        public $color='white';
    }
```

```
    $mao1=new mao();          //实例化猫类的一个对象
    //输出 mao1 对象的属性
    echo "mao1 的年龄为{$mao1->age}，体重为{$mao1->weight}，毛色为
    {$mao1->color}。<br />";
    $mao2=new mao();          //实例化猫类的一个对象
    //修改 mao2 对象的属性
    $mao2->age=2;
    $mao2->weight=1000;
    $mao2->color='black';
    //输出 mao2 对象的属性
    echo "mao2 的年龄为{$mao2->age}，体重为{$mao2->weight}，毛色为
    {$mao2->color}。<br />";
?>
```

代码运行结果如图 6.7 所示。

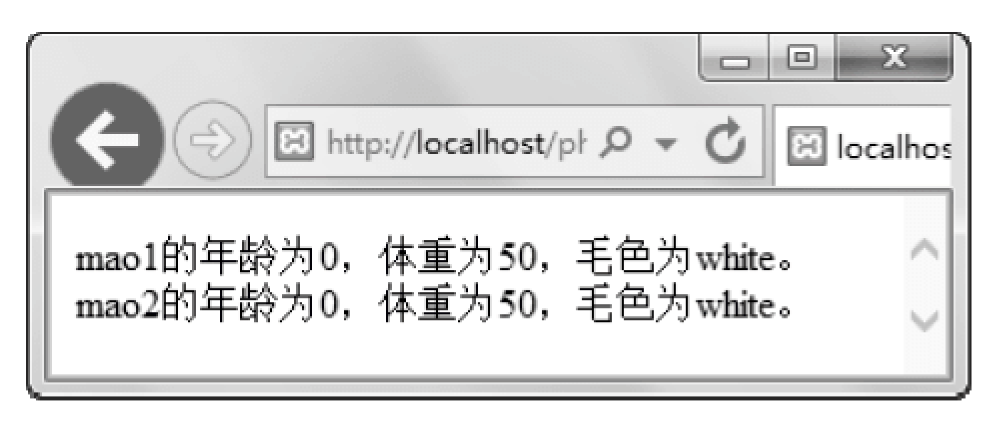



图 6.6　运行结果



图 6.7　运行结果

如图 6.7 所示即为默认对象和修改属性后对象的属性变化。

4. 对象之间是相互独立的

就同现实中一样，所有类的实例化即对象，均为独立的个体，任何一个对象的变化都不会影响到该类的其他对象。

【示例 6-9】 以下代码演示同一个类实例化的不同对象之间是相互独立的。

```
<?php
    class mao{                //定义猫类
        public $age=0;        //定义多个属性并初始化
        public $weight=50;
        public $color='white';
    }
    $mao1=new mao();          //实例化猫类的一个对象
    //输出 mao1 对象的属性
    echo "mao1 的年龄为{$mao1->age}，体重为{$mao1->weight}，毛色为
    {$mao1->color}。<br />";
    $mao2=new mao();          //实例化猫类的一个对象
    //修改 mao2 对象的属性
    $mao2->age=2;
    $mao2->weight=1000;
    $mao2->color='black';
    //输出 mao2 对象的属性
    echo "mao2 的年龄为{$mao2->age}，体重为{$mao2->weight}，毛色为
    {$mao2->color}。<br />";
    //再次输出 mao1 对象的属性
    echo "再次输出 mao1 对象的属性：的年龄为{$mao1->age}，体重为
    {$mao1->weight}，毛色为{$mao1->color}。";
?>
```

代码运行结果如图 6.8 所示。

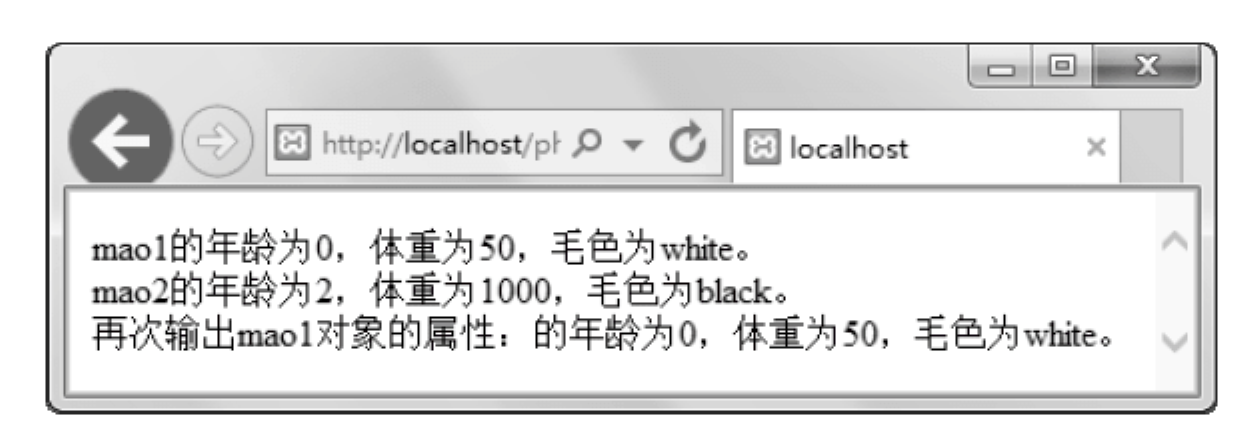


图 6.8 运行结果

从运行结果中可以看出，无论对象 mao2 的属性如何修改，均不会影响到 mao1 对象的属性。

5. 对象传递

在讲解实例化对象的知识的时候我们就知道，每个对象都有一个特定的标识符来表示。我们可以使用一个变量来保存这个标识符，以标识这个对象。这个变量可以为任何符合 PHP 命名规范的变量，也就是说，对象与变量名是无关的，那么我们就可以用多个变量来标识一个对象，就类似于给一个对象起了多个名字一样。

【示例 6-10】以下代码演示使用多个变量标识同一个对象。

```
<?php
    class mao{            //定义猫类
        public $age=0;    //定义多个属性并初始化
        public $weight=50;
        public $color='white';
    }
    $mao1=new mao();      //实例化一个猫类的对象
    $mao2=new mao();      //实例化一个猫类的对象
    $mao1_a=$mao1;        //变量$mao1_a 接受变量$mao1 的赋值，即标识同一个对象
    //输出各个变量的类型信息
    echo 'mao1:';
    var_dump($mao1);
    echo '<br />mao1_a:';
    var_dump($mao1_a);
    echo '<br />mao2:';
    var_dump($mao2);
?>
```

代码运行结果如图 6.9 所示。

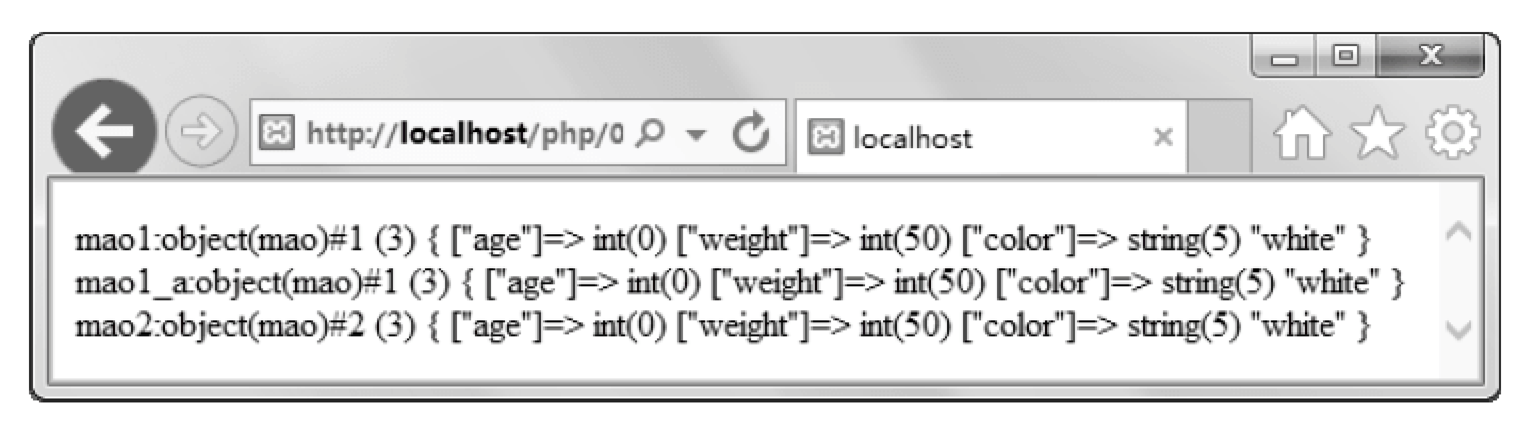


图 6.9 运行结果

注意：var_dump 输出表示对象的信息中，“#”后面的数字即代表该变量标识“#”前面“( )”中类的第几个对象。

从代码的运行结果可以看出，变量 mao1 和 mao1_a 均标识 mao 类的第一个对象，而

变量 mao2 则标识“mao”类的第二个对象。由于两个变量均标识同一个对象，因此使用其中任何一个变量访问对象所做的修改都会被保留。

【示例 6-11】以下代码演示使用标识同一个对象的变量，修改对象的属性并使用另一个标识该对象的变量输出。

```
<?php
    class mao{                      //定义猫类
        public $age=0;              //定义多个属性并初始化
        public $weight=50;
        public $color='white';
    }
    $mao1=new mao();                //实例化一个对象
    $mao1 a=$mao1;                  //将对象标识符赋值给另一个对象
    //使用变量$mao1 a 访问对象并修改其属性
    $mao1 a->age=3;
    $mao1 a->weight=1500;
    //使用变量$mao1 访问对象并输出其属性
    echo "mao1 的年龄为{$mao1->age}，体重为{$mao1->weight}，毛色为
    {$mao1->color}。"
?>
```

代码运行结果如图 6.10 所示。

从运行的结果就可以看出，使用标识同一个对象的任意一个变量对对象的修改均会被保留。

6. 类型运算符 instanceof

类型运算符 instanceof 用来确定一个 PHP 变量是否标识某一个类的对象，它的使用形式如下：

```
变量 instanceof 类名
```

如果“变量”是“类名”类的一个对象则表达式会返回 TRUE，否则返回 FALSE。

【示例 6-12】以下代码演示使用 instanceof 运算符，判断一个变量是否标识某一个类的对象。

```
<?php
    class ren{                      //定义人类

    }
    class mao{                      //定义猫类

    }
    $ren=new ren();                 //实例化一个人类的对象
    $mao=new mao();                 //实例化一个猫类的对象
    if($ren instanceof ren){        //判断变量$ren 是否标识人类对象
        echo '变量$ren 标识一个 ren 类对象。<br />';
    }else{
        echo '变量$ren 不是标识一个 ren 类对象。<br />';
    }
    if($mao instanceof ren){        //判断变量$mao 是否标识人类对象
        echo '变量$mao 标识一个 ren 类对象。';
    }else{
        echo '变量$mao 不是标识一个 ren 类对象。';
    }
?>
```

代码运行结果如图 6.11 所示。




图 6.10　运行结果



图 6.11　运行结果

从运行结果可以看到，使用类型运算符正确判断了变量是否标识指定类的对象。

7. 对象比较

同数组类似，PHP 中也有专门用来比较对象的运算符，运算符符号以及说明如表 6.2 所示。

表 6.2　对象比较运算符

运算符	说　　明
==	确定两个变量是否标识同一个类的对象，如是则返回 TRUE，否则返回 FALSE
!=	确定两个变量是否不标识同一个类的对象，如是则返回 FALSE，否则返回 TRUE
===	确定两个变量是否标识同一个类的同一个对象，如是则返回 TRUE，否则返回 FALSE
!==	确定两个变量是否不标识同一个类的同一个对象，如是则返回 FALSE，否则返回 TRUE

对象比较运算符的使用非常简单，我们直接通过一个示例来讲解。

【示例 6-13】以下代码演示对象比较运算符的使用方法及返回的结果。

```
<?php
    class ren{          //定义人类

    }
    class mao{          //定义人类

    }
    $ren=new ren();     //实例化人类的对象
    $ren_a=new ren();   //实例化人类的对象
    $ren1=$ren;         //对象传递
    $mao=new mao();     //实例化猫类的对象
    //输出变量的信息
    echo '$ren:';
    var_dump($ren);
    echo '<br />$ren_a:';
    var_dump($ren_a);
    echo '<br />$ren1:';
    var_dump($ren1);
    echo '<br />$mao:';
    var_dump($mao);
    //输出比较对象比较运算的结果
    echo '<br />******对象比较******<br />$ren==$ren_a:';
    var_dump($ren==$ren_a);
    echo '<br />$ren===$ren_a:';
    var_dump($ren===$ren_a);
    echo '<br />$ren1==$ren:';
    var_dump($ren1==$ren);
```

```
        echo '<br />$ren===$mao:';
        var_dump($ren1===$ren);
        echo '<br />$ren==$mao:';
        var_dump($ren==$mao);
        echo '<br />$ren1!=$mao:';
        var_dump($ren1!=$mao);
    ?>
```

代码运行结果如图 6.12 所示。



图 6.12　运行结果

读者可以对照每个对象的详细信息和对象比较的结果来理解对象比较运算符。

6.2.2　常量属性（类常量）

在 PHP 基础部分我们就学习过常量，通常不能随意改变的量就可以将它定义为常量。在一个类中，也可以将一些不能被随意改变的属性定义为常量。例如人的性别在出生后就确定，而且不能随意改变，我们就可以在定义类时将它定义为常量。

1. 定义类常量

在类中定义常量使用的是 const 关键字，而且在定义类常量的时候不需要使用“$”符号，它的使用形式如下：

```
const 常量名 = 常量值
```

常量值必须为一个定值，不能是变量，类属性或其他操作的结果。一个定义了类常量的类的示例如下：

```
class boy{
    const sex='男';
    public $age=15;
}
```

2. 访问类常量

与访问类中的变量不同的是，访问类中的常量使用的运算符是范围解析操作符（::）。类中的常量不仅可以通过该类的对象来访问，而且可以不实例化对象直接通过类名来访问。两种使用形式如下：

```
类名::类常量
```

```
对象::类常量
```

类常量的值在定义的时候已经确定，而且它的值不可再被更改。

【示例 6-14】以下代码演示在类中定义常量并且使用两种方式来访问它。

```
<?php
    class boy{                    //定义男孩类
        const sex='男';
        public $age=15;
    }
    echo '不实例化对象输出类常量：'.boy::sex;
    $boy=new boy();               //实例化男孩类的一个对象
    echo '<br />使用对象访问类常量：'.$boy::sex;
?>
```

代码运行结果如图 6.13 所示。



图 6.13　运行结果

从运行结果可以看出，使用两种形式均可以正确访问到类常量。读者一定要注意类常量的访问方式与类中的变量访问方式是不同的。

6.3　成 员 方 法

在类中成员方法就类似于该类所拥有的能力，而实质上就是类中定义的函数。类中的成员方法又分为普通成员方法和魔术方法，本节就来详细介绍这些知识。

6.3.1　普通成员方法

普通成员方法就是定义在类中的方法，通常可以通过对象来调用它，下面就来学习普通成员方法的知识。

1. 定义成员方法

类中的成员方法定义同普通函数定义相同，形式如下：

```
function 方法名(参数列表){
    方法语句;
}
```

不同之处就在于，在类中定义的成员方法也需要使用访问控制关键字来修饰以控制其可访问性。与成员属性不同的是，成员方法可以省略访问控制关键字，它的访问控制关键字会被系统设置为 public。因此以下两种定义成员方法的形式是等价的。

```
class 类名{
    public function 方法名(参数列表){
```

```
        方法语句;
    }
}

class 类名{
    function 方法名(参数列表){
        方法语句;
    }
}
```

其他两类成员方法形式如下：

```
class 类名{
    protected function 方法名(参数列表){
        方法语句;
    }
    private function 方法名(参数列表){
        方法语句;
    }
}
```

下面我们就来为人类定义一个 public 成员方法 walk，代码如下：

```
class ren{
    public function walk(){
        echo '我会走路。';
    }
}
```

2. 调用成员方法

因为成员方法也是类的成员，因此同样使用“->”符号来调用它，通常的形式如下：

```
对象 -> 方法名(参数列表)
```

同调用普通函数类似，在调用成员方法的时也必须按照成员方法定义时候的参数列传入对应的参数。

【示例 6-15】以下代码演示调用成员方法的语法以及调用成员方法后的效果。

```
<?php
    class ren{                          //定义人类
        public function walk(){         //定义人类的成员方法
            echo '我会走路。';
        }
    }
    $ren=new ren();                     //实例化一个人类的对象
    $ren->walk();                       //调用成员方法
?>
```

代码运行结果如图 6.14 所示。



图 6.14　运行结果

同成员属性类似，使用不同的访问控制标识符修饰的成员方法的可访问性都是不同的。

【示例 6-16】以下代码演示定义在人类中定义一个 private 修饰的成员方法，并尝试在类的外部访问它。

```
01  <?php
02      class ren{                              //定义人类
03          public function walk(){             //定义 public 成员方法
04              echo '我会走路。';
05          }
06          private function dance(){           //定义 private 成员方法
07              echo '我会跳舞。';
08          }
09      }
10      $ren=new ren();                         //实例化人类的对象
11      $ren->walk();                           //访问 public 成员方法
12      $ren->dance();                          //访问 private 成员方法
13  ?>
```

代码运行结果如图 6.15 所示。

从运行结果可以看到，当使用对象访问代码中第 3 行定义的 public 修饰的 walk 成员方法时，代码会正确运行。但是在使用对象访问代码中第 6 行定义的 private 修饰的 dance 成员方法时，就会报错，那是因为 private 修饰的成员不可以在类外部被访问。

同普通函数类似，类中的成员方法同样可以定义参数及默认参数。

【示例 6-17】以下代码演示为类中的成员方法定义参数以及默认参数。

```
01  <?php
02      class ren{                                      //定义人类
03          public function info($name,$age=3){         //定义有两个参数的成员方法
04              echo "我是{$name}，年龄是{$age}岁。<br />";
05          }
06      }
07      $jim=new ren();                                 //实例化人类的对象 jim
08      $jim->info('jim');                              //调用成员方法并传入参数
09      $tom=new ren();                                 //实例化人类的对象 tom
10      $tom->info('tom',10);                           //调用成员方法并传入参数
11  ?>
```

代码输出结果如图 6.16 所示。

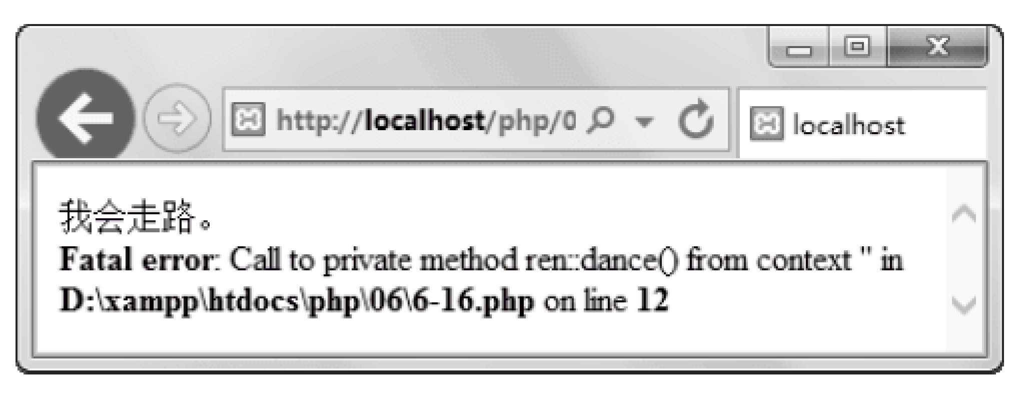


图 6.15　运行结果



图 6.16　运行结果

从代码第 8 和第 10 行的调用语句可以看出，成员方法的调用同普通函数的调用都是类似的，因此读者应该注意调用时传递不同个数的参数相应的返回结果。

3. 在成员方法中使用类成员

可能会有很大一部分读者就会疑惑了，使用 private 标识符描述的类成员既然不能在类的外部被访问，那它还有什么用处。其实通常情况下成员属性都会使用 private 或者 protected 来修饰，然后通过类中的成员方法来访问它们，这就使得这些属性同类外部形成一个保护屏障，也更加符合人类看待事物的规律。

例如我们的生日，如果我们自己不说出来那么别人就不会知道。这里的生日，我们就可以将它抽象为类中使用 private 修饰的成员属性，而说出来我们就可以将它抽象为类中使

用 public 修饰的成员方法。在类的成员方法中，访问该类中的成员属性需要使用“$this”伪变量。通常的使用形式如下：

```
class 类名{
    访问控制标识符 成员属性;
    访问控制标识符 funtion 方法名(参数列表){
        $this->成员属性名;
    }
}
```

按照上面的形式可以将前面所举的例子抽象为一个类，如下所示。

```
class ren{
    private $birthday='1990-12-20';
    public function say_birthday(){
        echo '我的生日是'.$this->birthday;
    }
}
```

为了让大家更好地理解“$this”这个伪变量的作用，我们首先来实例化两个人类的对象，代码如下：

```
$tom=new ren();
$jim=new ren();
```

我们知道，每个类的对象之间都是相互独立的，它们都有相同的成员。但是在每个对象中“$this”变量会根据所在的对象自动指代所在的对象，形式如图 6.17 所示。

```
人类的对象$jim
class ren{
    private $birthday='1990-12-20';
    public function say_birthday(){
        echo '我的生日是'.$jim->birthday;
    }
}
```

```
人类的对象$tom
class ren{
    private $birthday='1990-12-20';
    public function say_birthday(){
        echo '我的生日是'.$tom->birthday;
    }
}
```

图 6.17　$this 伪变量

图 6.17 就说明了$this 的作用，当然这些都是由 PHP 核心来完成的，我们并不需要去这样指定。

【示例 6-18】以下代码演示在类的成员方法中使用类的成员属性。

```
<?php
    class ren{                              //定义人类
        private $birthday='1990-12-20';     //定义 private 修饰的成员属性
        public function say_birthday(){     //定义 public 修饰的成员方法
            echo '我的生日是'.$this->birthday;
        }
    }
    $tom=new ren();                         //实例化人类的对象 tom
    echo 'tom: ';
    $tom->say_birthday();                   //访问对象的成员方法
    $jim=new ren();                         //实例化人类的对象 jim
    echo '<br />jim: ';
    $jim->say_birthday();                   //访问对象的成员方法
?>
```

代码运行结果如图 6.18 所示。

我们可以看到，程序调用类的成员方法成功输出了使用 private 修饰的成员属性。但是在很多读者看来，这并没有使得类的成员属性更加安全，反而好像多此一举。在成员方法中加入一些条件判断，即可真正提高成员属性的安全性。

【示例 6-19】以下代码演示在类的成员方法中，加入判断语句以控制对成员属性的访问。

```
<?php
    class ourself{                              //定义自己人类
        private $birthday='1990-12-20';         //定义 private 修饰的成员属性
        public function say_birthday($obj){//定义一个成员方法
            if($obj instanceof ourself){        //判断传入的参数是否为
                                                  ourself 类的对象
                echo '我的生日是'.$this->birthday;
            }else{
                echo '我不能告诉你我的生日。';
            }
        }
    }
    class other{                    //定义一个其他人类

    }
    $tom=new ourself();             //实例化自己人类的对象
    $jim=new ourself();             //实例化自己人类的对象
    $ken=new other();               //实例化其他人类的对象
    echo 'jim 想知道 tom 的生日：';
    $tom->say_birthday($jim);       //调用成员方法并将相同类的对象 jim 做为参数
    echo '<br />ken 想知道 tom 的生日：';
    $tom->say_birthday($ken);       //调用成员方法并将不同类的对象 ken 做为参数
?>
```

代码运行结果如图 6.19 所示。



图 6.18　运行结果



图 6.19　运行结果

从运行结果我们可以看到，当不同类的对象想要访问 tom 对象的生日属性时，根据成员方法的判断而做出不同的响应，而想要实现同样的效果，在不使用成员方法的情况下是不能实现的。

在类中，和变量属性对应的是常量属性，我们知道常量属性的访问方式与变量属性的访问方式是不同的。同样地，在成员方法中使用类常量需要使用伪变量 self。与伪变量“$this”类似，伪变量 self 会根据所在的类自动指代所在的类名，通常的使用形式如下：

```
class 类名{
    const 常量名=常量值;
    访问控制标识符 funtion 方法名(参数列表){
        self::常量名;
    }
```

```
}
```

由于常量在不需要实例化对象的情况下就可以被访问，因此我们就以一个简单的图来帮大家理解，如图 6.20 所示。

ren类

```
class ren{
    const name='ren';
    public function classname(){
        echo '这个类的名称：'.ren::name;
    }
}
```

mao类

```
class mao{
    const name='mao';
    public function classname(){
        echo '这个类的名称：'.mao::name;
    }
}
```

图 6.20　self 伪变量

当然图 6.20 所示的操作都是由系统来完成的，由于常量是不能被更改的，因此我们就以一个简单的示例来讲解这个知识。

【示例 6-20】以下代码演示在成员方法中访问类常量。

```
<?php
    class ren{                                      //定义人类
        const name='ren';
        public function classname(){
            echo '这个类的名称：'.self::name;       //访问类常量
        }
    }
    class mao{                                      //定义猫类
        const name='mao';
        public function classname(){
            echo '这个类的名称：'.self::name;       //访问类常量
        }
    }
    //实例化类的对象
    $ren=new ren();
    $mao=new mao();
    //访问成员方法
    echo '对象 ren 所在的类名为：';
    $ren->classname();
    echo '<br />对象 mao 所在的类名为：';
    $mao->classname();
?>
```

代码运行结果如图 6.21 所示。

使用 self 的优点就在于代码并不会随着类名的改变而出现错误。

我们知道成员方法也是类中的成员，因此类中的成员方法也可以使用类中的成员方法。与使用类的成员属性类似，使用类的成员方法的形式如下：

```
class 类名{
    访问控制标识符 funtion 方法名(参数列表){
        $this->成员方法名(参数列表);
    }
}
```

【示例 6-21】以下代码演示在类方法中调用类方法。

```
<?php
```

```
02      class ren{                                  //定义人类
03          private function dance(){               //定义 private 成员方法 dance()
04              echo '我要跳一支舞。';
05          }
06          private function sing(){                //定义 private 成员方法 sing()
07              echo '我要唱一首歌。';
08          }
09          public function do_something($item){    //定义 public 成员方法 do_something()
10              switch($item){
11                  case 'dance':
12                      $this->dance();             //调用类成员方法 dance()
13                      break;
14                  case 'sing':
15                      $this->sing();              //调用类成员方法 sing()
16                      break;
17              }
18          }
19      }
20      $ren=new ren();                             //实例化人类的对象
21      //访问类成员方法并传入不同的参数
22      $ren->do_something('sing');
23      $ren->do_something('dance');
24  ?>
```

代码运行结果如图 6.22 所示。



图 6.21　运行结果



图 6.22　运行结果

从运行的结果可以看到，代码中第 9 行定义的 do_something 成员方法根据传入的不同参数，分别调用了代码中第 3 行定义的 dance()成员方法和第 6 行定义的 sing()成员方法输出了不同的语句。

6.3.2　魔术方法

魔术方法是类中一些特殊的成员方法，之所以叫做魔术方法就是它们是比较神奇的。神奇之处就在于魔术方法是在指定的事件发生的时候会被系统自动执行而不需要人工去调用。魔术方法与普通成员方法的区别是使用双下划线（__）开头的。下面就来介绍一些常用的魔术方法。

1. 构造方法

构造方法是当一个类的对象被创建时进行一些操作。触发构造方法执行的事件是在实例化一个该类对象的时候。该函数的原型如下：

```
void __construct ([ mixed $args [, $... ]] )
```

可以看到构造方法可以定义多个参数。魔术方法都是由系统来调用的，那么我们就需

要了解怎么将参数传入到构造方法中。我们知道，构造方法是在创建对象的时候被触发的，首先来看实例化类的对象的代码：

```
变量=new 类名(参数列表)
```

在前面的学习过程中，一直没有用到的就是参数列表，这个参数列表用来为构造方法传递参数。构造方法通常用来对对象做一些初始化。

【示例 6-22】以下代码演示使用构造方法初始化类的对象。

```
01  <?php
02      class ren{                              //定义人类
03          private $name;                      //定义成员属性
04          public function __construct($name){//定义构造函数
05              $this->name=$name;
06          }
07          public function myname(){           //定义成员函数用于输出对象名称
08              echo '我的名字是：'.$this->name;
09          }
10      }
11      //实例化人类的对象
12      $tom=new ren('Tom');
13      $jim=new ren('Jim');
14      //访问对象的成员方法
15      $tom->myname();
16      echo '<br />';
17      $jim->myname();
18  ?>
```

代码运行结果如图 6.23 所示。

代码中第 12、13 行代码在实例化对象的时候进行了对象名称的初始化操作，因此在使用不同的对象调用成员方法时输出了不同的名字。当然构造方法不止用来做类似的事情，它还可以在类中执行一些操作。



图 6.23　运行结果

2. 析构方法

析构函数是与构造函数对应的魔术方法，它会在对象没有被索引和程序结束之前执行，它的函数原型如下：

```
void __destruct ( void )
```

析构函数不接受任何参数，在析构方法中通常是执行一些释放资源的操作。

【示例 6-23】以下代码演示在代码执行完毕后自动调用析构方法。

```
<?php
    class myclass{                  //定义一个类
        function __destruct(){      //定义析构方法
            echo '析构方法执行。<br />';
        }
    }
    //实例化类的两个对象
    $obj=new myclass();
    $obj2=new myclass();
?>
```

代码运行结果如图 6.24 所示。

结合代码和运行结果可以看出，虽然在代码中并没有销毁对象，但是在程序执行完毕后还是执行了析构方法，这是因为在对象不再使用的时候会被系统自动回收。

【示例 6-24】以下代码演示在执行代码时，显式地销毁对象索引时自动调用析构函数。

```
01  <?php
02      class myclass{                          //定义一个类
03          public function __destruct(){       //定义析构方法
04              echo '析构方法执行。<br />';
05          }
06      }
07      $myclass1=new myclass();                //实例化类的对象
08      echo '销毁对象与变量索引：';
09      unset($myclass1);                       //销毁变量$myclass1
10      $myclass2=new myclass();                //实例化类的对象
11      echo '程序执行完毕。'
12  ?>
```

代码运行结果如图 6.25 所示。



图 6.24　运行结果



图 6.25　运行结果

从运行结果可以看到，当程序在第 9 行代码调用 unset()函数销毁变量$myclass1 之后析构方法就被调用了。而变量$myclass 标识的对象则在代码执行完毕后系统回收对象时执行了析构方法。

3. __get()、__set()、__isset()、__unset()和__call()

除了构造方法和析构方法这两个魔术方法之外，PHP 中还有一些常用的魔术方法，它们的原型如下：

```
public void __set(string $name,mixed $value)
public mixed __get(string $name)
public bool __isset(string $name)
public void __unset(string $name)
public mixed __call(string $name,array $arguments)
```

这些方法的作用如下。

- __set(string $name, mixed $value)：在对象给未定义或者不可见的属性赋值时被调用，参数 name 即为需要赋值的属性，value 即为需要为属性赋的值。
- __get(string $name)：在对象访问未定义或者不可见的属性时被调用，参数 name 即为需要访问的属性名。
- __isset(string $name)：对未定义或者不可见的属性使用 isset()时被调用，参数 name 即为属性名。
- __unset(string $name)：对未定义或者不可见的属性使用 unset()时被调用，参数 name

即为属性名。

- __call(string $name,array $arguments)：在对象调用未定义或者不可见的方法时被调用，参数 name 即为方法名，参数 arguments 为要传递给方法的参数组成的数组。

综上所述，以上这些模式方法的作用就是拦截到访问不存在的对象成员的操作，然后对其做相应的响应，并且这些方法都必须使用 public 修饰。在进行示例演示之前，我们首先需要了解它们的运行方式。首先，定义一个类并实例化该类的一个对象，如下所示。

```
class ren{
    public   set($name,$value){

    }
    public   get($name){

    }
}
$ren=new ren();
```

以上定义了一个类并实例化了一个对象，当执行如下语句的时候：

```
$ren->age=15;
```

由于我们想要赋值的属性并不存在，所以会被“__set()”方法拦截到，并且会得到相应的参数，如下所示。

```
public __set('age',15){

}
```

然后就可以对其进行相应的操作。类似的情况，当我们执行如下的语句时：

```
$ren->sex;
```

由于对象试图访问的属性并不存在，因此会被“__get()”方法拦截到，并且获得相应的参数，如下所示。

```
public __get('sex'){

}
```

其他的魔术方法的运行方式都与“__set()”和“__get()”魔术方法类似，这里就不再详细列举。

【示例 6-25】以下代码演示使用“__set()”魔术方法，处理为不存在的对象属性赋值的操作。

```
<?php
    class ren{          //定义一个人类
        //定义成员属性
        private $name='tom';
        private $age=15;
        //定义__set()魔术方法
        public function __set($name,$value){
            echo "你不能为不存在或不可访问的{$name}属性赋值。<br />";
        }
    }
    $ren=new ren();     //实例化人类的对象
    $ren->sex='boy';    //为不存在的变量赋值
```

```
    $ren->name='jim';    //为不可访问的变量赋值
?>
```

代码运行结果如图 6.26 所示。

结合代码和运行结果可以看出，在对象不存在或者为不可访问的对象属性赋值时，就可以被“__set()”魔术方法拦截并处理。接下来我们来看一段使用“__get()”成员方法的最简单示例。

【示例 6-26】以下代码演示使用“__get()”魔术方法获取对象的成员属性。

```
<?php
    class ren{          //定义人类
        //定义成员属性
        private $name='Tom';
        private $age=15;
        //定义__get()魔术方法
        public function __get($name){
            return $this->$name;
        }
    }
    $ren=new ren();        //实例化人类的一个对象
    //输出人类的私有成员属性
    echo $ren->name;
    echo $ren->age;
?>
```

代码运行结果如图 6.27 所示。



图 6.26　运行结果

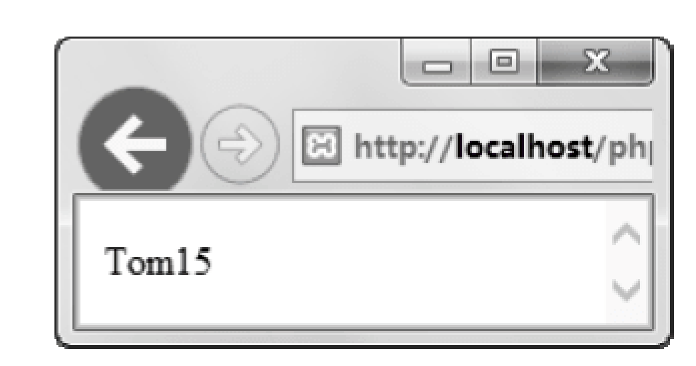


图 6.27　运行结果

结合代码和运行结果可以看到，通常在类外部不可访问的 private 描述的成员属性均通过“__get()”魔术方法被返回。很显然，类似的操作破坏了对象的机制，因为这会使得 private 失去期望的作用。下面我们就来演示一段实用的“__get()”魔术方法的使用方式。

【示例 6-27】以下代码演示实用的“__get()”魔术方法使用方式。

```
<?php
    class ren{                                    //定义人类
        //定义成员属性
        private $name='Tom';
        private $age=15;
        //定义成员方法
        public function getname(){
            return $this->name;
        }
        public function getage(){
            return $this->age;
        }
        //定义__get()魔术方法
        public function __get($name){
            $method="get{$name}";                 //定义变量用来接收方法名称
```

```
            if(method_exists($this,$method))//判断类中是否存在指定的方法
                return $this->$method();
            else
                echo "<br />你不可以获取不存在或者不可访问的{$name}成员属性。";
        }
    }
    $ren=new ren();                                    //实例化一个对象
    //访问存在的成员属性
    echo '输出对象的名字：'.$ren->name;
    echo '<br />输出对象的年龄：'.$ren->age;
    echo $ren->sex;                                    //访问不存在的成员属性
?>
```

代码运行结果如图 6.28 所示。

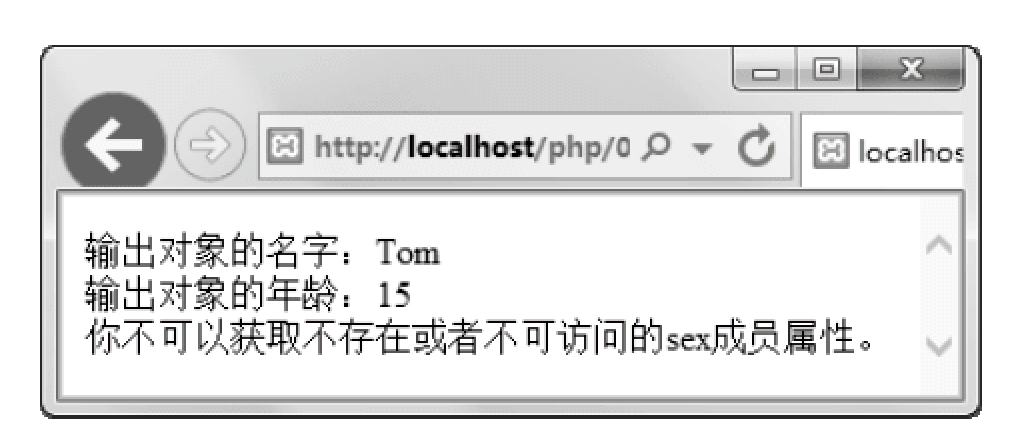


图 6.28　运行结果

从代码中可以看到，“__get()”魔术方法中通过调用已经存在的成员方法，从而保护了 private 标识符的作用。其他魔术方法的使用同“__set()”和“__get()”方法类型，这里就不再详细介绍。

6.4　类的继承

继承是面向对象编程的特性之一。形象地来讲就类似于父亲和孩子的关系。孩子会遗传一些父亲的特性，而且孩子也可以有自己特性，当然孩子也可以从父亲那里学到一些能力并将其改进。继承的作用就是使得代码可复用性得到很大的提高。下面我们就通过代码来实现以上的情况，进而轻松学习类的继承知识。

6.4.1　成员访问标识符

在前面的学习过程中，我们就已经在使用成员访问控制标识符，它们分别是 private、protected 和 public。它们在类定义中用来限制类成员的可访问性。在类的继承中，它们同时又有不同的特点：

- 使用 public 和 protected 修饰的类成员均可以被继承；
- 使用 private 修饰的类成员不可以被继承。

在类继承中使用的关键字是 extends，它的使用形式如下：

```
class 继承类 extends 被继承类{

}
```

在以上的语法中，继承类被称为被继承类的子类；被继承类称为继承类的父类。下面演示简单的继承：

```
class class1{

}
class class2 extends class1{

}
```

在以上的代码中，class2 为 class1 的子类；class1 为 class2 的父类。

1. public 标识符

使用 public 标识符修饰的类成员均可以被子类继承。这里我们就拿父子来做举例，public 成员属性就类似于父亲有两只眼睛和两只耳朵，而且会走路一样。虽然这看起来是再平常不过的事情，但是对于严谨的编程语言来说，就需要这样严谨的定义。

【示例 6-28】以下代码演示儿子类继承父亲类的 public 修饰的成员。

```
01  <?php
02      class father{                       //定义 father 类
03          //定义 public 描述的成员属性和方法
04          public $eyes=2;
05          public $ears=2;
06          public function walk(){
07              echo '走路。';
08          }
09      }
10      class son extends father{           //定义 son 类并且继承自 father 类
11
12      }
13      $father=new father();               //实例化一个 father 类的对象
14      //访问 father 类的属性和方法
15      echo "father 有{$father->eyes}个眼睛，有{$father->ears}个耳朵并且
                                                                    father 会";
16      $father->walk();
17      $son=new son();                     //实例化 son 类的对象
18      //访问 son 类的属性和方法
19      echo "<br />son 有{$son->eyes}个眼睛，有{$son->ears}个耳朵并且 son 会";
20      $son->walk();
21  ?>
```

代码运行结果如图 6.29 所示。



图 6.29　运行结果

从代码中可以看出，虽然在 son 类中并没有定义任何类成员，但在代码第 19、20 行访问相应的类成员时均可以正确访问到，这就是类继承的作用。现在的 son 类的代码就类似于如下所示：

```
class son{
        public $eyes=2;
        public $ears=2;
```

```
        public function walk(){
            echo '走路。';
        }
    }
```

2. private 标识符

使用 private 修饰的类成员不可以被子类继承。我们同样拿父子来举例，private 修饰的属性就类似于父亲有天生的卷发，有吸烟的习惯，而这些特性都不让儿子继承。

【示例 6-29】以下代码演示 private 标识符修饰的类成员不可以被继承。

```
<?php
    class father{                          //定义 father 类
        //定义 private 修饰的类成员和方法
        private $hair='curly hair';
        private function smoke(){
            echo '我有吸烟的习惯。';
        }
    }
    class son extends father{              //定义继承自 father 类的 son 类
        //定义访问 private 修饰的类成员的方法
        public function get_property(){
            $this->hair;
            $this->smoke();
        }
    }
    $son=new son();                        //实例化 son 类的对象
    $son->get_property();                  //调用对象的方法
?>
```

代码运行结果如图 6.30 所示。



图 6.30　运行结果

从运行结果可以看出，在代码中试图访问$hair 成员属性和 smoke()成员方法的时候均报错，其中的原因就在于这些 father 类中的成员均不可以被 son 类继承。现在的 son 类的代码就类似于如下所示。

```
class son{

}
```

3. protected 标识符

从前面的 public 和 private 标识符的介绍中我们可以得知，public 修饰的成员可以被子类继承，而且其成员可以在任何地方被访问；而 private 修饰的成员只可以在类内部被访问而且不可以被子类继承。在有些情况下需要一类成员，需要它们不可以在类外部被访问到，

但是可以被继承。protected 修饰符就是为了满足这个需求而产生的。同样用父子来做比喻的话，这就类似于父亲有一笔财产和一门独家的手艺，儿子可以继承它们，但是外人不会知道儿子继承了多少财产和学到了什么手艺。

接下来就来演示 protected 标识符的特点和作用，但是为了代码可以正确执行完毕，我们在类中定义了魔术方法来处理访问不存在或者不可见的操作，当然这些魔术方法也会被子类继承。

【示例 6-30】以下代码演示 protected 标识符的特点及作用。

```
<?php
    class father{                     //定义 father 类
        //定义 protected 修饰的成员属性和方法
        protected $money=1000000;
        protected function cook(){
            echo '烹饪。';
        }
        //定义魔术方法
        public function __call($name,$arr){
            echo '你要访问的类方法不存在或者不可以访问。<br />';
        }
        public function __get($name){
            echo '你要访问的类属性不存在或者不可以访问。<br />';
        }
    }
    class son extends father{         //定义继承自 father 类的 son 类
        //定义类方法以访问从父类继承到的 protected 修饰的成员
        public function getinfo(){
            echo "我得到了{$this->money}财产，并且我学会了";
            $this->cook();
        }
    }
    //实例化 father 类并访问其成员
    $father=new father();
    $father->money;
    $father->cook();
    //实例化 son 类并访问其成员
    $son=new son();
    $son->money;
    $son->cook();
    $son->getinfo();
?>
```

代码运行结果如图 6.31 所示。

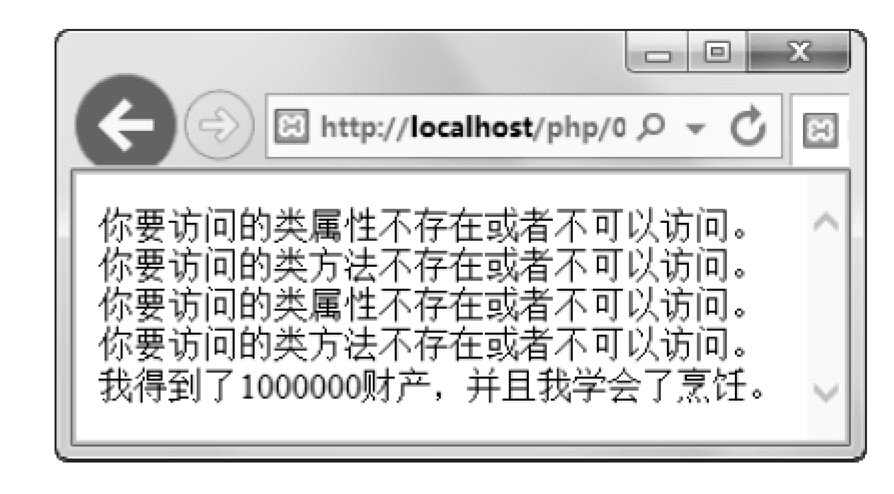


图 6.31　运行结果

从运行结果可以看出，在类外访问父类和子类的成员均不会成功，但是通过在子类中访问 protected 修饰的类成员还是正确地被输出。由此可见，father 类中 protected 修饰的成员确实被 son 类所继承。现在的 son 类的代码就类似于如下所示。

```
class son{
        protected $money=1000000;
        protected function cook(){
            echo '烹饪。';
```

```
        }
        public function   call($name,$arr){
            echo '你要访问的类方法不存在或者不可以访问。<br />';
        }
        public function   get($name){
            echo '你要访问的类属性不存在或者不可以访问。<br />';
        }
    }
```

4. 在子类中添加成员方法

从上面的讲解中可以看到，继承可以很大限度地提高代码的可复用性。继承不仅仅只可以继承父类的方法，而且还可以在子类中添加自己的方法。例如以上的示例中，儿子类可以继承父类的 cook 成员方法从而拥有了烹饪的能力。在子类中添加成员方法就类似于儿子又自学了一种能力，从而很方便地扩展了子类的功能。

【示例 6-31】以下代码演示在子类中定义 programme()方法。

```
<?php
    class father{                               //定义 father 类
        public function cook(){
            return '烹饪';
        }
    }
    class son extends father{                   //定义 son 类
        public function programme(){            //定义儿子类的 programme 成员方法
            return '编程';
        }
    }
    $son=new son();                             //实例化 son 类的一个对象
    //访问儿子对象的成员方法
    echo '儿子会'.$son->cook().'和'.$son->programme();
?>
```

代码运行结果如图 6.32 所示。

结合代码和运行结果可以看出，son 类使用继承的方式很容易就将父类进行了功能扩展，这也是面向对象编程的一个重要的功能。现在的 son 类的代码就类似于如下所示。

```
class son extends father{
        public function cook(){
            return '烹饪';
        }
        public function programme(){
            return '编程';
        }
    }
```

为子类扩展一个功能的代码我们已经简单地实现了，扩展多个功能的方式都与之类似，这里就不再详细介绍。

5. 在子类重写父类的成员方法

在前面的学习中，我们为子类添加了自己的成员方法，但是有些时候是需要基于父类的成员方法扩展的，也就是说要使用同一个方法名。要实现这个功能，只需要在子类中定义同父类相同的方法名但不同的实现即可。

【示例 6-32】以下代码演示在子类中重写父类的方法。

```
<?php
    class father{                          //定义 father 类
        public function cook(){
            return '烹饪';
        }
    }
    class son extends father{              //定义继承在 father 类的 son 类
        public function cook(){
            return '高级烹饪';
        }
    }
    $father=new father();                  //实例化 father 类的一个对象
    echo '父亲会'.$father->cook();         //访问对象的成员方法
    $son=new son();                        //实例化 son 类的一个对象
    echo '<br />儿子会'.$son->cook();      //访问对象的成员方法
?>
```

代码运行结果如图 6.33 所示。



图 6.32　运行结果



图 6.33　运行结果

从运行结果可以看到，在子类中重写父类的方法后就会执行子类定义的方法而不会报错。需要注意的是，在子类中重写父类的方法时，方法的访问权限要与父类中方法的访问权限相同或者更为宽松，准确地描述即为：

- ❑ 父类中的成员方法访问标识符为 public，则子类中重写此方法必须定义访问标识符为 public。
- ❑ 父类中的成员方法访问标识符为 protected，则子类中重写此方法必须定义访问标识符为 protected 或者 public。
- ❑ 父类中的成员方法访问标识符为 private，则子类中重写此方法必须定义访问标识符为 private。

【示例 6-33】以下代码演示子类重写父类方法时访问控制符的定义。

```
<?php
    class father{                          //定义 father 类
        //定义 protected 成员方法
        protected function cook(){
            return 'protected cook';
        }
        protected function programme(){
            return 'protected programme';
        }
        protected function kung fu(){
            return 'private kung fu';
        }
    }
    class son extends father{
        protected function cook(){         //重写父类的 cook 成员方法，访问
                                             控制符为 protected
```

```
16              return 'protected cook';
17          }
18          public function programme(){     //重写父类的 programme 成员方法，
                                               访问控制符为 public
19              return 'public programme';
20          }
21          private function kung fu(){      //试图重写父类的 kung fu 成员方法，
                                                访问控制符为 private
22              return 'public kung fu';
23          }
24      }
25  ?>
```

代码运行结果如图 6.34 所示。



图 6.34　运行结果

从运行结果可以看出，当代码第 21～23 行试图重写父类中的成员方法时代码会报错，其中的原因就在于，子类中重写父类方法时使用了比父类 protected 访问标识符更为严格的 private 访问标识符。读者在编码过程中一定要注意。

6. 在子类中访问父类的成员方法

前面学习了在子类中重写父类的成员方法，实质上并没有真正将它重写，父类的方法还存在于子类中，只是在调用的时候系统会自动决定调用子类中重写的方法。这里我们将要学习的就是在子类中调用被系统隐藏的父类方法。调用的语法非常简单，如下所示。

```
parent::方法名(参数列表)
```

在子类中访问父类的方法是非常有用的，特别是在有构造函数的类中，在子类中调用父类的构造方法可以保证初始化工作都正确完成。

【示例 6-34】以下代码演示在子类中访问父类的成员方法。

```
<?php
    class father{                         //定义 father 类
        public function method(){         //定义方法
            echo '<br />father method';
        }
    }
    class son extends father{             //定义继承自 father 类的 son 类
        public function method(){         //重写父类方法
            echo 'son method';
        }
        public function parent_method(){          //定义方法
            parent::method();             //调用父类方法
        }
    }
    $son=new son();                       //实例化 son 类的对象
```

```
        //调用 son 类的方法
        $son->method();
        $son->parent_method();
?>
```

代码运行结果如图 6.35 所示。



图 6.35　运行结果

从运行结果就可以看到，在子类中仍然成功调用了父类中的方法。

6.4.2　final 关键字

final 关键字是在 PHP 中新加入的。在有的情况下，我们希望一个类或者方法完成一个确定的功能，为了防止破坏这个功能，就可以将类或者方法使用 final 修饰。修饰类的语法形式如下：

```
final class 类名{

}
```

使用 final 关键字修饰的类不可以被继承。

【示例 6-35】以下代码演示使用 final 关键字修饰的类不可以被继承。

```
<?php
    final class final_class{                //定义 final 修饰的类

    }
    class myclass extends final_class{      //试图继承 final 修饰的类，程序会报错

    }
?>
```

代码运行结果如图 6.36 所示。

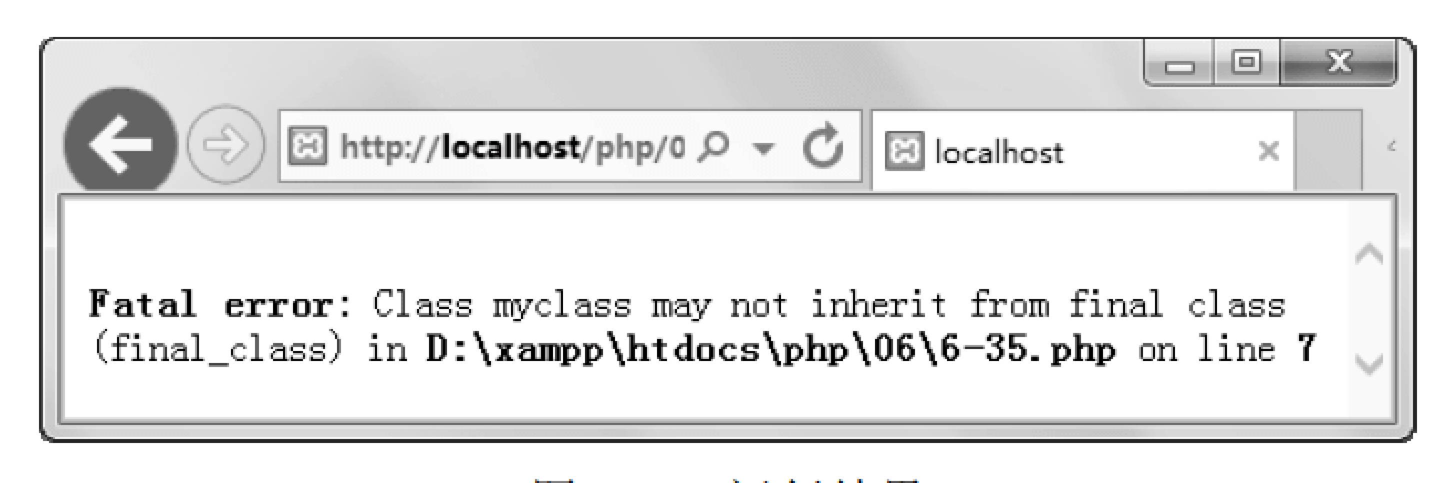


图 6.36　运行结果

从运行结果可看出，当 myclass 类试图继承 final_class 类时程序会报错。

修饰类方法的语法如下：

```
class 类名{
    final 访问控制标识符 function 方法名(参数列表){
```

```
        方法体;
    }
}
```

使用 final 关键字修饰的类方法不可以被子类重写。

【示例 6-36】以下代码演示使用 final 关键字修饰类方法。

```
<?php
    class myclass{                      //定义一个类
        final public function myfunc(){//定义 final 修饰的类方法
            return 'final function';
        }
    }
    class test extends myclass{         //定义继承自 myclass 类的 test 类
        public function myfunc(){       //试图从父类继承的 myfunc 类方法，
                                          程序会报错
            return 'my function';
        }
    }
?>
```

代码运行结果如图 6.37 所示。

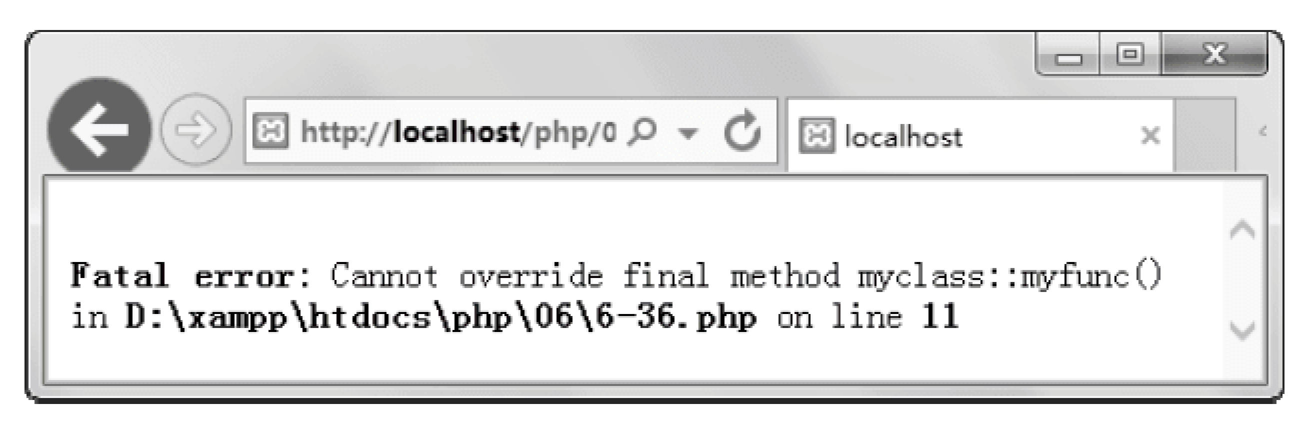


图 6.37　运行结果

从运行结果可以看出，当子类中试图重写父类中使用 final 关键字修饰的成员方法时，会导致程序出错。

6.4.3　static 关键字

static 关键字可以用来修饰类的成员和方法，还可以用来完成后期静态绑定的功能。下面就来介绍这些知识。

1. 修饰类成员

使用 static 关键字修饰的类成员的一个特点是不需要经过实例化一个对象就可以访问，这在有些情况下是非常有用的。例如为了查看类的一些信息，我们就可以直接通过类名来访问，因而省去了实例化对象的开销。另一个特点是使用 static 修饰的成员属性会保存操作的结果。实例化类成员的形式如下：

```
访问标识符 static 类成员（成员属性或方法）
```

需要注意的是，使用 static 关键字修饰的成员属性不可以使用对象访问，而成员方法是可以通过对象访问的。static 修饰的类成员均使用“::”来访问，不可用对象通过“->”来访问。

【示例 6-37】以下代码演示使用类名访问类方法获取类的信息。

```
01  <?php
02      class test{                                    //定义一个类
03          public static function class_info(){       //定义类方法
04              return '这是一个用于测试的类。';
05          }
06      }
07      echo test::class_info();        //使用类名访问 static 修饰的类方法
08  ?>
```

代码运行结果如图 6.38 所示。

从代码运行结果可以看出，在代码第 7 行使用类名就很容易地访问到了类的成员方法，从而获取到该类的信息。

【示例 6-38】以下代码演示使用 static 关键字修饰的类属性的特点及访问类属性的方法。

```
01  <?php
02      class test{                               //定义一个类
03          public static $num=0;                 //定义 static 修饰的类属性
04          public function __construct(){        //定义构造方法
05              self::$num++;                     //每次调用均使$num 变量递增
06          }
07      }
08      //实例化 test 类的 3 个对象
09      $test1=new test;
10      $test2=new test;
11      $test3=new test;
12      //输出 test 类实例化对象的个数
13      echo 'test 类一共实例化了'.test::$num.'个对象。';
14  ?>
```

代码运行结果如图 6.39 所示。



图 6.38　运行结果



图 6.39　运行结果

从运行结果可以看出，在代码第 5 行的每次递增都被保存了下来，从而使得在代码第 13 行访问该属性时输出 3。

当然，使用 static 修饰的类成员依然遵循访问标识符的规定，也就是说，即使一个成员属性是由 static 修饰的，但它的访问标识符为 private，那么它也不能在类外部被访问。同样，使用 static 修饰的成员方法依然遵循访问标识符的规定。

2. 后期静态绑定

在讲解本节的知识前，我们需要首先来看一个示例。

【示例 6-39】以下代码期望使用继承的构造方法访问各个类中对应的成员方法。

```
<?php
    class test{                           //创建 test 类
        public function __construct(){
            self::getinfo();              //调用类的成员方法
```

```
05          }
06          public static function getinfo(){
07              echo '实例化一个 test 类的对象<br />';
08          }
09      }
10      class test1 extends test{       //创建继承自 test 类的 test1 类
11          public static function getinfo(){
12              echo '实例化一个 test1 类的对象<br />';
13          }
14      }
15      class test2 extends test1{      //创建继承自 test1 类的 test2 类
16          public static function getinfo(){
17              echo '实例化一个 test2 类的对象<br />';
18          }
19      }
20      //实例化 3 个类的对象
21      $test=new test();
22      $test1=new test1();
23      $test2=new test2();
24  ?>
```

代码运行结果如图 6.40 所示。

从以上运行结果可以看出，示例中的第 4 行代码并没有分别调用子类中定义的成员方法。这种执行效果是由 PHP 系统决定的，这是由于 PHP 代码在编译的阶段就已经确定了“self::”所关联的成员方法。因此当类的构造函数被调用时，总会调用父类中的成员方法，这就使得一些功能无法被实现。PHP 为了实现我们期望的功能，使用 static 关键字来实现“后期静态绑定”，即代码会在运行时判断应该调用哪个成员。

【示例 6-40】以下代码演示使用“static”关键字实现后期静态绑定的功能。

```
01  <?php
02      class test{                     //创建 test 类
03          public function __construct(){
04              static::getinfo();      //后期静态绑定
05          }
06          public static function getinfo(){
07              echo '实例化一个 test 类的对象<br />';
08          }
09      }
10      class test1 extends test{       //创建继承自 test 类的 test1 类
11          public static function getinfo(){
12              echo '实例化一个 test1 类的对象<br />';
13          }
14      }
15      class test2 extends test1{      //创建继承自 test1 类的 test2 类
16          public static function getinfo(){
17              echo '实例化一个 test2 类的对象<br />';
18          }
19      }
20      //实例化 3 个类的对象
21      $test=new test();
22      $test1=new test1();
23      $test2=new test2();
24  ?>
```

代码运行结果如图 6.41 所示。

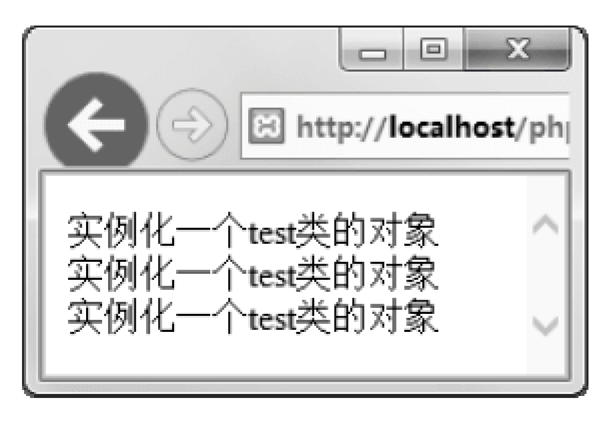



图 6.40　运行结果



图 6.41　运行结果

从以上运行结果可以看出，代码输出了我们所期望的结果。

6.5　面向对象高级使用

在前面的学习过程中，我们已经讲解了类的大部分知识，本节我们将要讲解的是一些比较特殊的类和一些使用方法。我们已经学习过使用 final 关键字修饰的特殊类，这里就不再讲解。本节主要介绍抽象类和接口，以及一些其他使用方法。

6.5.1　抽象类

抽象类的产生是由于有一些方法不能在父类中具体实现，就需要在子类中重写，但是有可能会在子类中遗漏，因此使用 abstract 关键字来强制子类必须实现。使用 abstract 关键字修饰的成员方法称为抽象方法，语法形式如下：

```
abstract 访问标识符 function 方法名(参数列表);
```

注意：抽象方法不能有具体的实现。

在类中如果至少有一个成员方法被声明为抽象方法，那么这个类必须使用 abstract 关键字修饰，这种类被称为抽象类，形式如下：

```
abstract class 类名{
    //至少有一个抽象方法
}
```

抽象类不能被直接实例化，它首先需要子类继承并实现抽象方法，然后再实例化子类。如果继承一个抽象类而不实现类中定义的抽象方法，那么程序会报错。

【示例 6-41】以下代码演示子类不实现抽象类中的抽象方法，程序会报错。

```
<?php
    abstract class father{                      //定义一个抽象类
        abstract public function test();        //定义抽象方法
    }
    class son extends father{
            //子类并未实现父类中定义的抽象方法
    }
?>
```

代码运行结果如图 6.42 所示。

从运行结果可以看出，程序报错并提示我们实现抽象方法或者将类声明为抽象类。这也就又引出了抽象类的另一个知识点，就是在子类中如果仍然不实现父类中的抽象方法，那么可以将这个子类再次声明为抽象类。

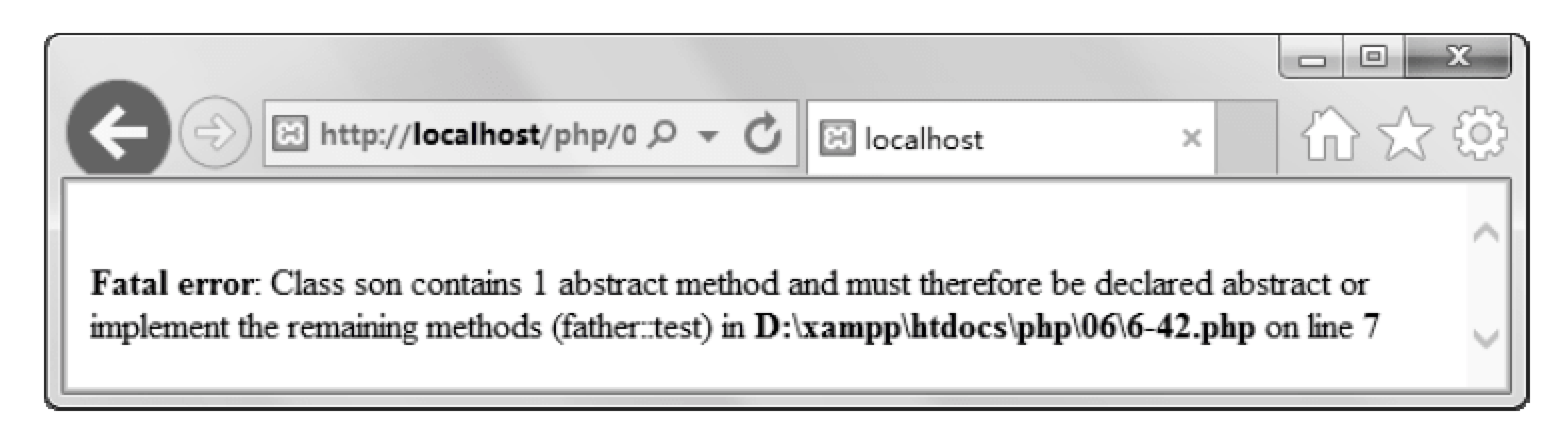


图 6.42　运行结果

【示例 6-42】以下代码演示子类继承抽象父类并再次声明为抽象类。

```
<?php
    abstract class father{                          //定义一个抽象类
        abstract public function test();            //定义抽象方法
    }
    abstract class son extends father{
            //子类并未实现父类中定义的抽象方法
    }
?>
```

代码运行结果如图 6.43 所示。

从运行结果可以看出，代码在运行后并没有报错，但是当我们要继承 son 类的时候，仍然需要实现类中的抽象方法或者再次声明为抽象类。

在继承类的方法时，方法的访问权限要与父类中方法的访问权限相同或者更为宽松，这同重写父类方法的要求是相同的。

【示例 6-43】以下代码演示在子类中实现父类的抽象方法。

```
<?php
    abstract class father{                          //定义一个抽象类
        abstract protected function test();         //定义抽象方法
    }
    class son1 extends father{                      //son1 类继承 father 类
        protected function test(){
            //实现部分
        }
    }
    class son2 extends father{                      //son1 类继承 father 类
        public function test(){
            //实现部分
        }
    }
?>
```

注意：子类实现父类的抽象方法的方法体可以为空，但是不可以省略花括号。

代码运行结果如图 6.44 所示。



图 6.43　运行结果



图 6.44　运行结果

运行结果没有错误输出，证明了程序中第 6 行和第 11 行的实现方式是正确的。需要注意的是，如果抽象方法在定义时指定了参数，那么在子类中实现的时候必须使用相同个数的参数。

【实例 6-44】 以下代码演示在子类中实现指定参数的抽象方法。

```
01  <?php
02      abstract class father{
03          abstract public function getname($name);    //定义抽象方法
04      }
05      class son1 extends father{
06          public function getname($name){             //正确的实现
07
08          }
09      }
10      class son2 extends father{
11          public function getname($myname){           //正确的实现
12
13          }
14      }
15      class son3 extends father{
16          public function getname($name,$age){  //错误的实现，因为参数个数不同
17
18          }
19      }
20  ?>
```

代码运行结果如图 6.45 所示。

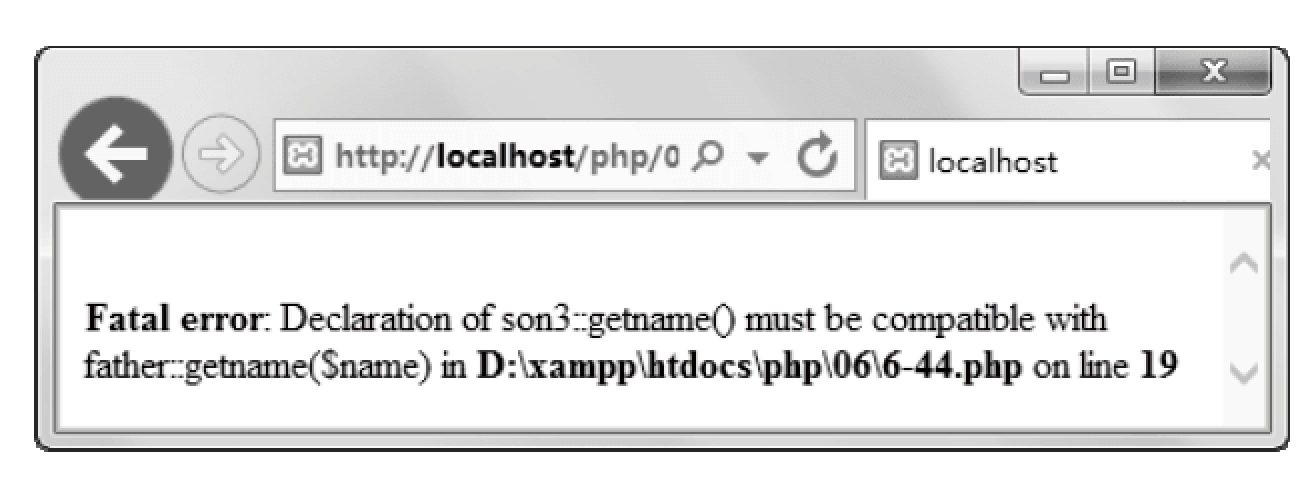


图 6.45　运行结果

我们可以看到程序会在运行时出现错误，原因就在于第 16 行代码实现抽象方法时的参数个数不同。抽象类中也可以定义普通的类成员。

【示例 6-45】 以下代码演示在抽象类中定义普通的成员。

```
<?php
    abstract class ren{          //定义人类
        //定义成员属性
        protected $name='';
        protected $age=0;
        //定义成员方法
        public function __construct($name,$age){
            $this->name=$name;
            $this->age=$age;
        }
        //定义抽象方法
        abstract public function getinfo();
    }
    class boy extends ren{       //继承 ren 类并实现其抽象方法
        public function getinfo(){
```

```
            return $this->name;
        }
    }
    class girl extends ren{     //继承 ren 类并实现其抽象方法
        public function getinfo(){
            return $this->age;
        }
    }
    //实例化两个子类的对象
    $boy=new boy('Tom',13);
    $girl=new girl('Mary',12);
    //调用对象的成员方法
    echo '男孩的名字为：'.$boy->getinfo();
    echo '<br />女孩的年龄为：'.$girl->getinfo();
?>
```

代码运行结果如图 6.46 所示。

以上代码中，两个子类以不同的实现代码实现了父类中的抽象方法。



图 6.46　运行结果

6.5.2　接口

接口可以用来规定一个类必须实现哪些方法，它的所有成员方法均为抽象方法并且可访问性必须为 public。

1. 接口的基础知识

定义接口的关键字为 interface，语法如下：

```
interface 接口名{
    (public) 方法名(参数列表);
    …
}
```

两个接口示例如下：

```
interface setinfo{
    public function setname();
    public function setage();
    public function settel();
}
interface getinfo{
    public function getname();
    public function getage();
    public function gettel();
}
```

实现接口使用的关键字为 implements，使用形式如下：

```
class 类名 implements 接口名{

}
```

实现一个接口的类，在类中必须实现接口中的所有抽象方法，否则程序会报错。

【示例 6-46】以下代码演示未全部实现接口中的抽象方法会出现的错误。

```
<?php
    interface setinfo{       //定义一个接口
        public function setname();
        public function setage();
        public function settel();
    }
    class info implements setinfo{
        public function setname(){
            //实现部分
        }
        public function setage(){
            //实现部分
        }
        //未实现 settel 魔术方法
    }
?>
```

代码运行结果如图 6.47 所示。

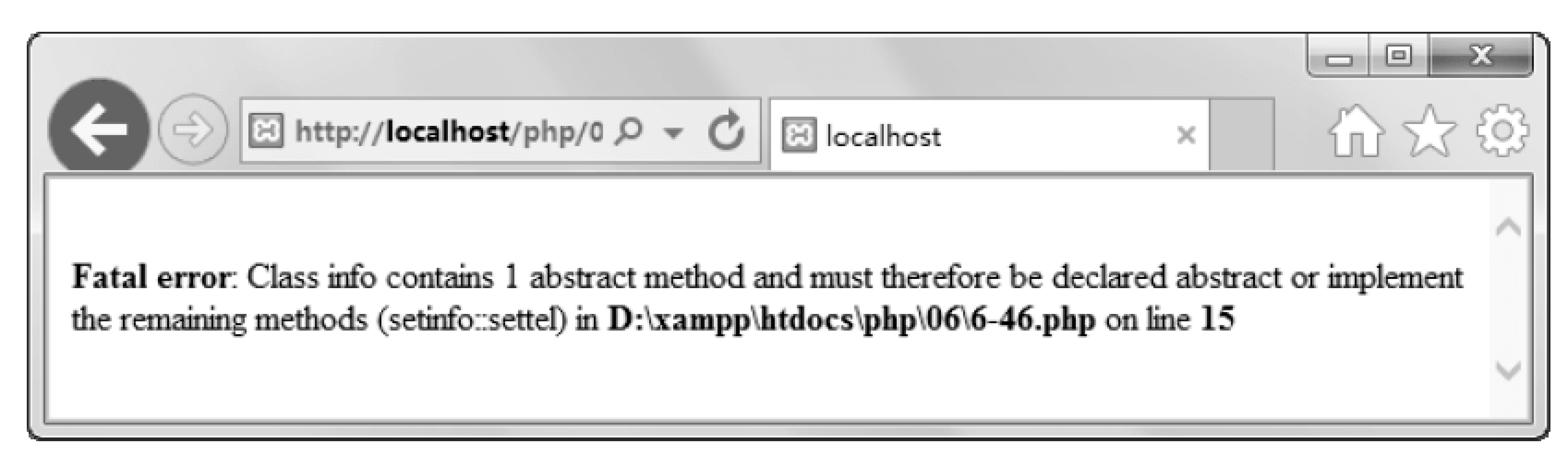


图 6.47　运行结果

从运行结果可以看出，程序运行后报错，同时也提示我们要实现剩下的方法或者将这个类声明为魔术类。

与魔术方法不同的是，一个类可以同时实现多个接口，多个接口名称之间需要使用逗号分隔，使用形式如下：

```
class 类名 implements 接口名,接口名,…{

}
```

【示例 6-47】以下代码演示一个类同时实现两个接口。

```
<?php
    interface setinfo{                          //定义一个接口
        public function setname();
        public function setage();
        public function settel();
    }
    interface getinfo{                          //定义一个接口
        public function getname();
        public function getage();
        public function gettel();
    }
    class info implements setinfo,getinfo{      //同时实现两个接口
        public function setname(){
            //实现部分
        }
        public function setage(){
            //实现部分
        }
```

```
        public function settel(){
            //实现部分
        }
        public function getname(){
            //实现部分
        }
        public function getage(){
            //实现部分
        }
        public function gettel(){
            //实现部分
        }
    }
?>
```

代码运行结果如图 6.48 所示。

由于在本段代码中并没有任何输出，因此运行结果会为空，由于我们实现了两个接口中的全部抽象方法，因此程序不会报错。

在实现多个接口时，需要注意的是这些接口中不应该有同名的抽象方法，这样虽然不会导致程序报错，但是会导致一个接口的功能不能被实现，读者在实际使用中一定要注意。

【示例 6-48】以下代码演示实现两个有同名方法的接口。

```
<?php
    interface setinfo{                          //定义一个接口
        public function setname();
        public function setage();
    }
    interface set_info{                         //定义一个接口
        public function setname();
    }
    class info implements setinfo,set_info{     //实现两个接口
        public function setname(){
            //实现部分
        }
        public function setage(){
            //实现部分
        }
    }
?>
```

代码运行结果如图 6.49 所示。

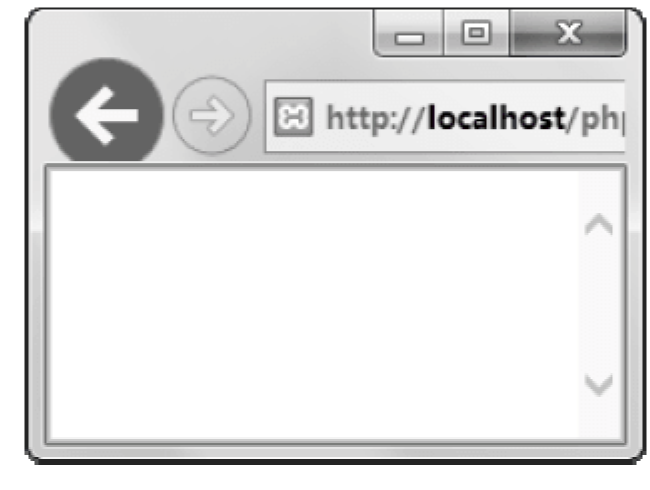


图 6.48　运行结果

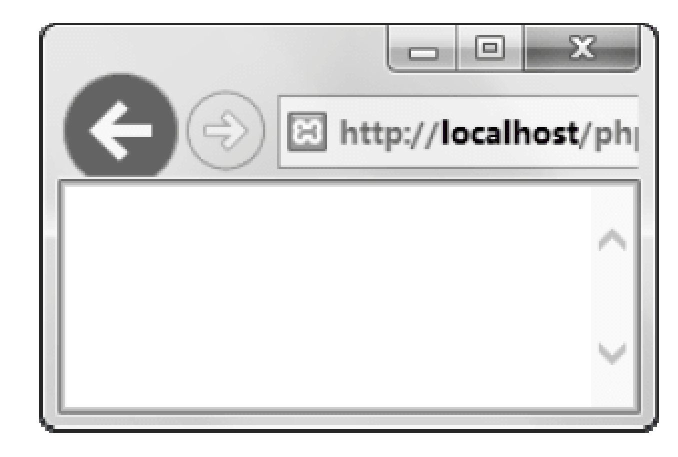


图 6.49　运行结果

从运行结果可以看出，程序并没有报错。但是该类实现了两个接口，而两个接口中抽象方法一共为 3 个，而我们在该类中只能实现两个类，也就是说总会有接口的功能没有被实现，这是一定要注意的。

同普通类类似，接口也可以通过 extends 关键字实现继承。

【示例 6-49】以下代码演示接口的继承。

```
<?php
    interface get_method{        //定义一个接口
        public function getname();
        public function getage();
    }
    interface getinfo extends get_method{
                                 //继承 get_method 接口并加入一个新的接口
        public function gettel();
    }
    class info implements getinfo{
        //该类中必须实现 3 个魔术方法 getname，getage 和 gettel
    }
?>
```

代码运行结果如图 6.50 所示。

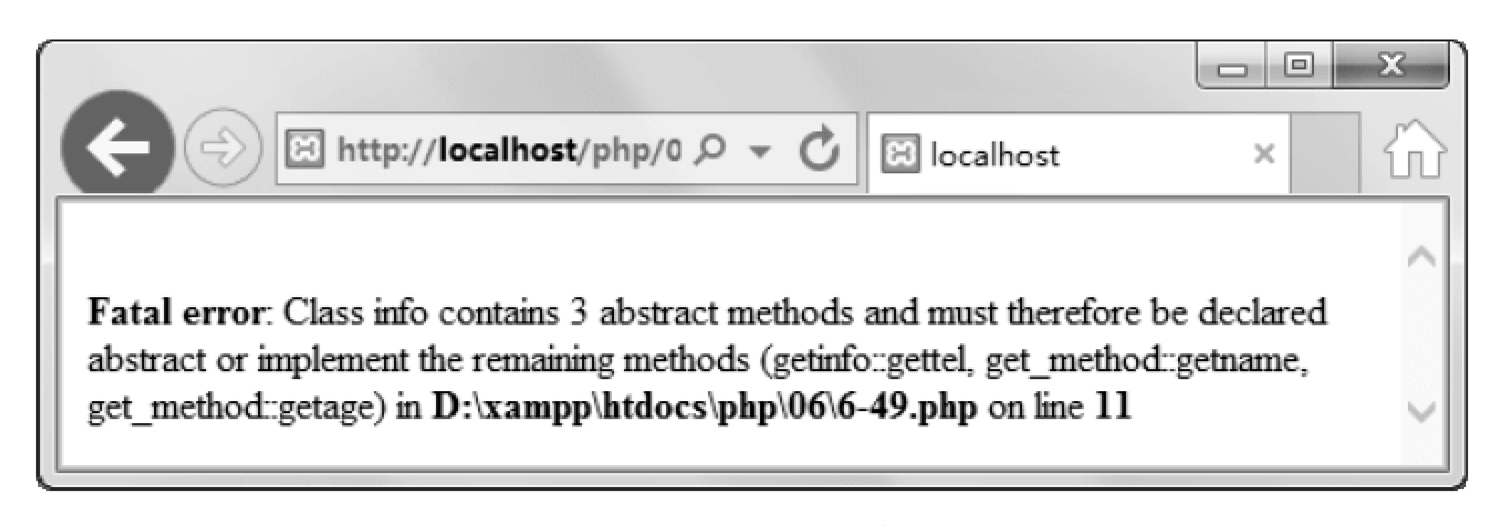


图 6.50　运行结果

从运行结果我们可以验证继承确实被成功执行，这在需要扩展一个接口的时候是非常有用的。

在接口类中除了可以定义抽象方法之外，还可以定义常量，接口常量的使用方法与普通类相同，这里就不再详细介绍。

2. 接口的应用

前面几乎介绍了全部的接口语法知识点，下面我就来介绍接口在实际中的作用。接口在多人合作开发的项目中尤其重要。例如多人合作开发一个项目，其中有多个程序员负责开发一个类似秒表的计时程序模块，如果没有为他们定义一个接口，那么程序员 A 就需要定义一个如下的类来实现这个模块。

```
class timer{
    public function start(){
        //实现部分
    }
    public function stop(){
        //实现部分
    }
}
```

程序员 B 可能会这样实现：

```
class timer{
    public function on(){
        //实现部分
```

```
    }
    public function off(){
        //实现部分
    }
}
```

程序员 C 又可以这样实现：

```
class timer{
    public function go(){
        //实现部分
    }
    public function stop(){
        //实现部分
    }
}
```

这样做导致的结果就是 3 个程序员所做的模块都有 3 种不同的调用方法，当其他人需要使用这些模块时就不得不去了解 3 个人所做的模块的调用方法。在做比较大的项目时，通常模块都由不同的人来开发。使用接口就可以规范这些开发，而具体的实现规则由各个程序员自己发挥。例如，我们就可以为这个定时模块定义一个如下接口：

```
interface timer{
    public function start();
    public function stop();
}
```

在各个程序员编写模块时，首先实现这个接口然后各自实现具体的内容。当别人需要调用这个模块时，就可以直接通过对象来调用这些统一的方法。

【示例 6-50】 以下代码演示使用接口实现约束两个类的优势。

```
<?php
    interface timer{                    //定义接口
        public function start();
        public function stop();
    }
    class a implements timer{           //a 类实现接口
        public function start(){
            echo 'A 运行';
        }
        public function stop(){
            echo 'A 停止';
        }
    }
    class b implements timer{           //b 类实现接口
        public function start(){
            echo 'B 运行';
        }
        public function stop(){
            echo 'B 停止';
        }
    }
    //实例化两个类的对象
    $a=new a();
    $b=new b();
    //定义一个调用的函数
    function control($obj){
```

```
27          if($obj instanceof timer){  //判断对象是否为 timer 类的实例
28              //分别调用指定的方法
29              $obj->start();
30              $obj->stop();
31          }
32      }
33      //调用函数并传入不同的对象
34      control($a);
35      control($b);
36  ?>
```

代码运行结果如图 6.51 所示。

从运行结果可以知道，程序正确调用了两个对象的成员方法。如果没有接口来约束成员方法的名称，那么就不可能使用第 26 行所定义的函数来调用各个对象的成员方法。



图 6.51　运行结果

6.5.3　其他使用

除了以上两种常用的使用方法之外，还有一些其他的使用方法我们也需要了解。它们分别是类型约束、traits 和设计模式，下面我们就来简单学习一下。

1. 类型约束

类型约束的作用就是约束传入成员方法中的参数类型，它可以约束为某个类的对象，在后来的版本中又加入了可以约束为一个数组。通常的使用形式如下：

```
访问控制符 function 方法名(类型约束 参数,类型约束 参数…){
    //实现语句
}
```

一个简单的示例代码如下：

```
class control{
    public function controller(timer $obj){
        $obj->start();
        $obj->stop();
    }
}
```

如果传入的参数不为约束的类，则程序运行会报错。同样地，如果约束为数组，则传入的参数不为数组，程序运行就会报错。

【示例 6-51】以下代码演示约束一个成员方法的参数。

```
<?php
    class timer{                        //定义 timer 类
        public function start(){
            echo '定时器开始! ';
        }
        public function stop(){
            echo '定时器停止! ';
        }
    }
    class control{                      //定义 control 类
        static function controller(timer $obj){//类型约束为 timer 类的对象
            //调用对象的成员方法
```

```
13              $obj->start();
14              $obj->stop();
15          }
16      }
17      class other{                          //定义一个 other 类
18          public function   construct(){  //定义构造函数
19              echo '这是一个其他类。';
20          }
21      }
22      control::controller(new timer);     //调用类的成员方法并传入正确的参数
23      control::controller(new other);     //调用类的成员方法并传入错误的参数
24  ?>
```

代码运行结果如图 6.52 所示。



图 6.52　运行结果

我们可以看到运行结果报错了，错误的原因就是在代码的第 23 行传入的参数不是约束的类型。

2. traits

traits 是 PHP 实现的一种代码复用的方法，它可以将一些类成员放入一个命名的结构中，在类中可以通过使用这个结构来复用那些代码。Traits 的使用和理解都非常简单，它的定义方法如下：

```
trait 名称{
    //成员
}
```

一个简单的示例如下：

```
trait info{
    static function getinfo(){
        return '这是一个'.__CLASS__.'类。<br />';
    }
}
```

使用这个结构的方法如下：

```
class 类名{
    use Trait 名称;
}
```

一个简单的使用示例如下：

```
class car{
    use info;
}
```

以上的代码就类似于如下代码：

```
class car{
    static function getinfo(){
        return '这是一个'.__CLASS__.'类。<br />';
    }
}
```

这样就可以使用简单的一段代码来代替很多的代码，使得程序更加简洁。

【示例 6-52】以下代码演示 traits 的简单使用。

```
<?php
    trait info{             //定义 trait
        static function getinfo(){
            return '这是一个'.__CLASS__.'类。<br />';
        }
    }
    class ren{              //定义 ren 类
        use info;           //使用 trait
    }
    class mao{              //定义 mao 类
        use info;           //使用 trait
    }
    class car{              //定义 car 类
        use info;           //使用 trait
    }
    //使用类名调用类的静态成员方法获得类的信息
    echo ren::getinfo();
    echo mao::getinfo();
    echo car::getinfo();
?>
```

代码运行结果如图 6.53 所示。



图 6.53　运行结果

通过以上代码可以看到，使用 traits 可以很大程度地减少代码的重复。

6.6　小　　结

面向对象编程是当前比较热门的方向，在本章中我们尽可能地介绍了 PHP 所有的面向对象的编程的知识。按照面向对象编程的思想来说，它更加符合人类看待事物的规律，因此理解起来并不会特别的困难。读者需要加强的是抽象的能力，这在面向对象编程中是比较重要的。

6.7　本 章 习 题

1. 定义一个 dog 类，该类拥有毛色（color）、体型（somatotype）公有属性和 run 公有方法。

2. 为 dog 类定义构造函数，在实例化对象的同时为属性毛色（color）和体型（somatotype）赋值。

3. 实例化 dog 类的一个对象，并访问该对象的公有属性毛色（color）、体型（somatotype）和公有方法 run，该方法需要返回“飞奔”。输出效果应类似图 6.54 所示。

图 6.54　运行效果

4. 定义一个继承自 dog 类的 siberian_husky 类，该类有 gnar（吠）公有方法，调用该方法应该返回“狂吠不止”。

第 7 章　错 误 处 理

错误在编程的过程中通常是无法避免的，我们在前面的学习过程中也碰到了多处错误。PHP 系统可以帮助我们提示以及修正一些错误，我们也可以自己定义一些抛出和处理错误方法。本章将详细讲解错误发送的原因、种类和如何处理这些错误。

7.1　错误发生的原因

错误发生的原因有多种，它们按照特点可以分为 4 大类，分别是语法错误、环境错误、逻辑错误和运行时错误。本节就来介绍这些错误。

7.1.1　语法错误

语法错误是最普遍也是最好解决的一类错误。顾名思义，语法错误就是由于程序员在程序编写的过程中使用了不能被 PHP 解析的错误语法。下面就来介绍一些常见的语法错误。

1. 语句未正确结束

这类错误通常是一条语句缺少了语句结束符号——分号。这类错误通常根据错误提示即可找到大概的位置并修正。

【示例 7-1】以下代码演示语句未正确结束的错误。

```
01  <?php
02      $num=10
03      echo $num;
04  ?>
```

代码运行结果如图 7.1 所示。

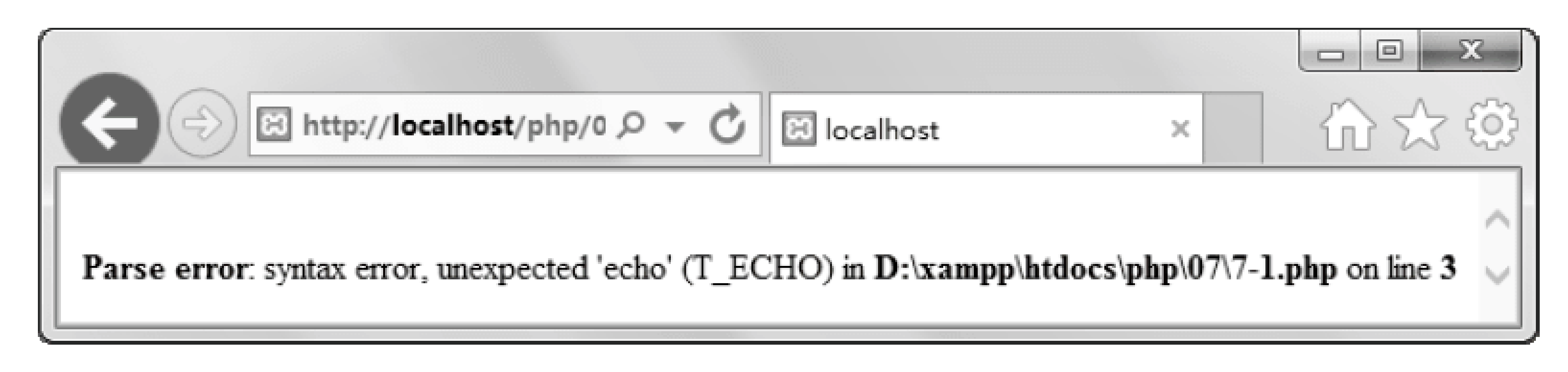


图 7.1　运行结果

从运行结果可以看出，程序在运行后报错，提示在代码第 3 行遇到了意料之外的 echo，而真正的原因是我们在代码的第 2 行没有正确地结束。解决办法就是在第 2 行加入结束符即可，修正后的代码如下：

```
01  <?php
02      $num=10;
```

```
03      echo $num;
04  ?>
```

2. 括号不匹配

这种错误通常出现在多层嵌套和判断条件比较长的代码中。

【示例 7-2】以下代码演示括号不匹配的错误。

```
01  <?php
02      if((($a&&$b)||($b&&$c)){
03          return 0;
04      }
05  ?>
```

代码运行结果如图 7.2 所示。



图 7.2　运行结果

从运行后的错误提示可以看到，代码中第 2 行出现了不期望的“{”，其实我们的错误是在 if 后多了一个“(”。修正后的代码如下：

```
01  <?php
02      if(($a&&$b)||($b&&$c)){
03          return 0;
04      }
05  ?>
```

这种错误通常可以通过编写代码时输入成对的括号后再在括号中加入代码避免。

【示例 7-3】以下代码演示花括号不匹配的错误。

```
01  <?php
02      class car
03          public function test(){
04              return 'test function.';
05          }
06      }
07  ?>
```

代码运行结果如图 7.3 所示。



图 7.3　运行结果

从运行后的错误提示可以看出，代码第 3 行中出现了不期望的 public，而期望的是“{”，我们只需根据提示在第 2 行代码后加入“{”即可。修正后的代码如下：

```
01  <?php
```

```
    class car{
        public function test(){
            return 'test function.';
        }
    }
?>
```

3. 错误的关键字

这种错误在没有关键字提示的编辑器中出现很频繁，而且错误的提示也常常会使人摸不着头脑。

【示例 7-4】以下代码演示拼写错误的关键字引发的错误。

```
<?php
    fucntion test(){
        return '这是一个测试函数。';
    }
?>
```

代码运行结果如图 7.4 所示。

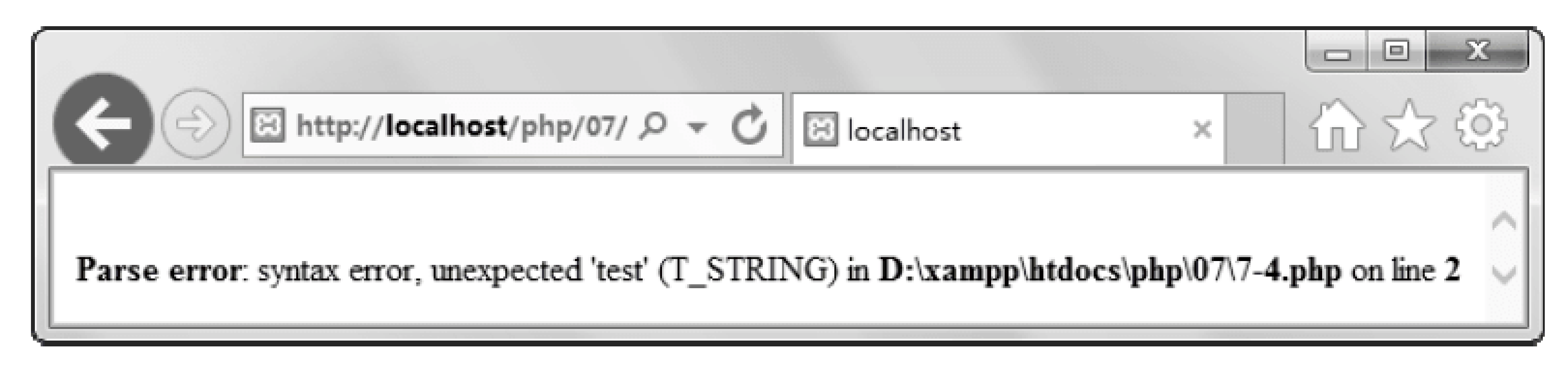


图 7.4　运行结果

运行结果提示我们有不期望的 test，而真正的原因是 fucntion 拼写错误，初次遇到这种提示有时会使人很费解。修正后的代码如下：

```
<?php
    function test(){
        return '这是一个测试函数。';
    }
?>
```

语法错误的种类还有很多，这里不再一一列举。

7.1.2　环境错误

环境错误通常是指 PHP 运行的环境和相关服务关联的问题，例如 PHP 程序需要使用的相关模块没有被正确加载、服务器没有启动、数据库配置错误等。由于我们使用的是集成开发环境，因此这些问题出现的概率是比较小的，集成环境的制作者已经替我们做了大量的工作。但是如果是自己初次搭建一个环境来运行，通常情况下会非常吃力。环境错误所涉及的编程语言之外的知识比较多，因此本书我们不做详细讲解。

7.1.3　逻辑错误

逻辑错误要比我们前面介绍的两种错误要隐蔽。逻辑错误的代码在运行时不会出现明显的错误提示，但是其运行的结果却并不是我们所期望的。

【示例 7-5】以下代码演示有逻辑错误的代码。

```
01 <?php
02     function compare($i,$j){        //定义一个比较函数
03         if($i>$j){
04             echo "<br />\$i={$i},\$j={$j}:{$i}>{$j}";
05         }elseif($i=$j){
06             echo "<br />\$i={$i},\$j={$j}:{$i}={$j}";
07         }elseif($i<$j){
08             echo "<br />\$i={$i},\$j={$j}:{$i}<{$j}";
09         }
10     }
11     //调用函数进行比较
12     compare(25,10);
13     compare(30,30);
14     compare(25,30);
15     compare(1,99);
16 ?>
```

代码运行结果如图 7.5 所示。

从运行结果可以看出，程序没报出任何的错误，而且单从运行结果来看也找不出任何的错误。但是仔细看程序代码中的第 14 和第 15 行调用函数时传入的参数，然后再对照运行结果来看，就会发现我们期望比较的是 25 和 30、1 和 99 的大小，但是结果却成为了 30 和 30、99 和 99 来比较大小。而代码的错误就是第 5 行中我们将比较运算符（==）写成了赋值运算符（=）。修正代码后的运行结果如图 7.6 所示。

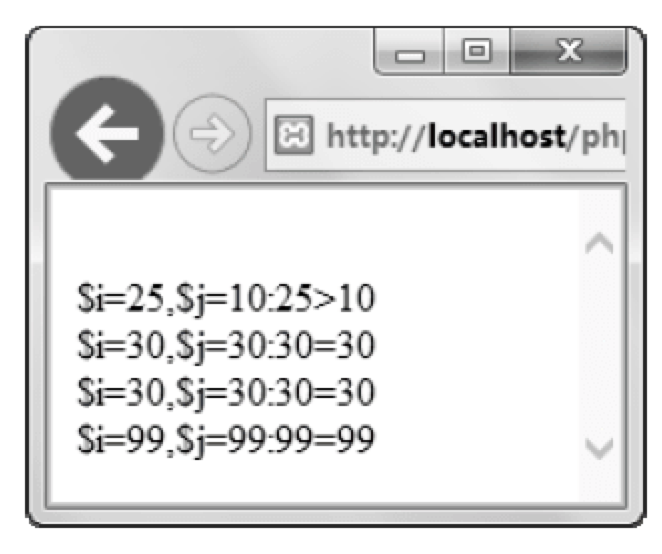


图 7.5　运行结果

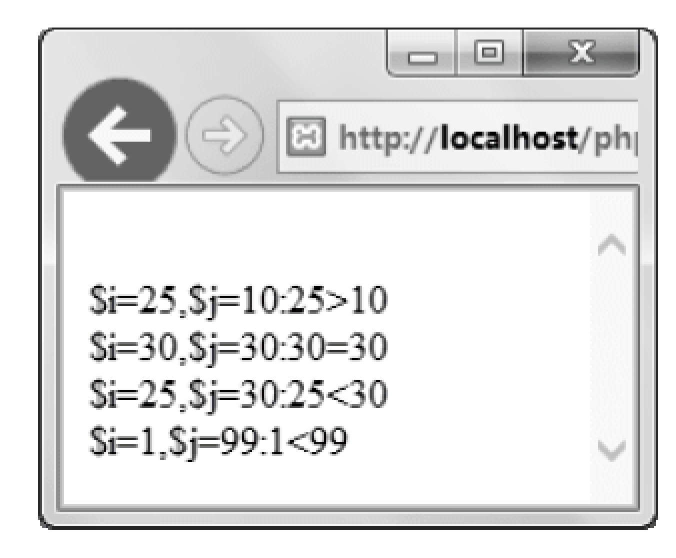


图 7.6　修正代码后的运行结果

这种错误往往就需要靠经验和使用工具来修正了。

7.1.4　运行时错误

运行时错误是在程序运行过程中出现的。有这种错误的代码往往在语法上没有任何错误，这种错误出现的情况一般是不能正确地使用到资源。例如，要打开的文件不存在或者没有操作的权限、连接数据库的账号或者密码错误等。这种错误还是比较容易解决的，例如，可以在读取文件之前首先检查文件是否存在等措施。

7.2　错误的分类

PHP 中的错误分为标准错误和异常，下面就来介绍标准错误和异常。

7.2.1　预定义错误常量

PHP 中定义了一些预定义的错误常量来表示需要采取动作的错误等级，详细的内容如

表 7.1 所示。

表 7.1　预定义错误常量

值	常　　量	说　　明
1	E_ERROR	致命的运行时错误。这类错误一般是不可恢复的情况，后果是导致脚本终止不再继续运行
2	E_WARNING	运行时警告（非致命错误）。仅给出提示信息，但是脚本不会终止运行
4	E_PARSE	编译时语法解析错误
8	E_NOTICE	运行时通知。表示脚本遇到可能会表现为错误的情况，但是在可以正常运行的脚本里面也可能会有类似的通知
16	E_CORE_ERROR	在 PHP 初始化启动过程中发生的致命错误
32	E_CORE_WARNING	PHP 初始化启动过程中发生的警告（非致命错误）
64	E_COMPILE_ERROR	致命编译时错误
128	E_COMPILE_WARNING	编译时警告（非致命错误）
256	E_USER_ERROR	用户产生的错误信息。由用户自己在代码中使用 PHP 函数 trigger_error()来产生的
512	E_USER_WARNING	用户产生的警告信息。由用户自己在代码中使用 PHP 函数 trigger_error()来产生的
1024	E_USER_NOTICE	用户产生的通知信息。由用户自己在代码中使用 PHP 函数 trigger_error()来产生的
2048	E_STRICT	启用 PHP 对代码的修改建议，以确保代码具有最佳的互操作性和向前兼容性
4096	E_RECOVERABLE_ERROR	可被捕捉的致命错误。它表示发生了一个可能非常危险的错误，但是还没有导致 PHP 引擎处于不稳定的状态
8192	E_DEPRECATED	运行时通知。启用后将会对在未来版本中可能无法正常工作的代码给出警告
16384	E_USER_DEPRECATED	用户产生的警告信息。由用户自己在代码中使用 PHP 函数 trigger_error()来产生的
30719	E_ALL	E_STRICT 除外的所有错误和警告信息

这些常量以及值，可以用来在 PHP 配置文件中配置需要采取动作的错误等级，也可以使用 error_reporting()函数来指定当前程序需要采取动作的错误等级。这些常量可以使用为操作符来实现组合使用。

7.2.2　错误提示配置

错误提示配置也就是指定 PHP 系统对那些错误采取动作。在前面我们也提到过，错误配置有两种方式，一种是通过配置 PHP 配置文件来完成，一种为使用 error_reporting()函数来完成。下面就来分别介绍这两种方式。

1. php.ini 文件

php.ini 文件是 PHP 的配置文件，它用来设置 PHP 系统执行的功能及加载的模块和相关模块的设置。XAMPP 集成环境的 php.ini 文件在安装主文件夹（xampp）下的 php 文件

夹中。我们可以从控制面板打开主文件夹后找到它。操作流程如图 7.7 所示。

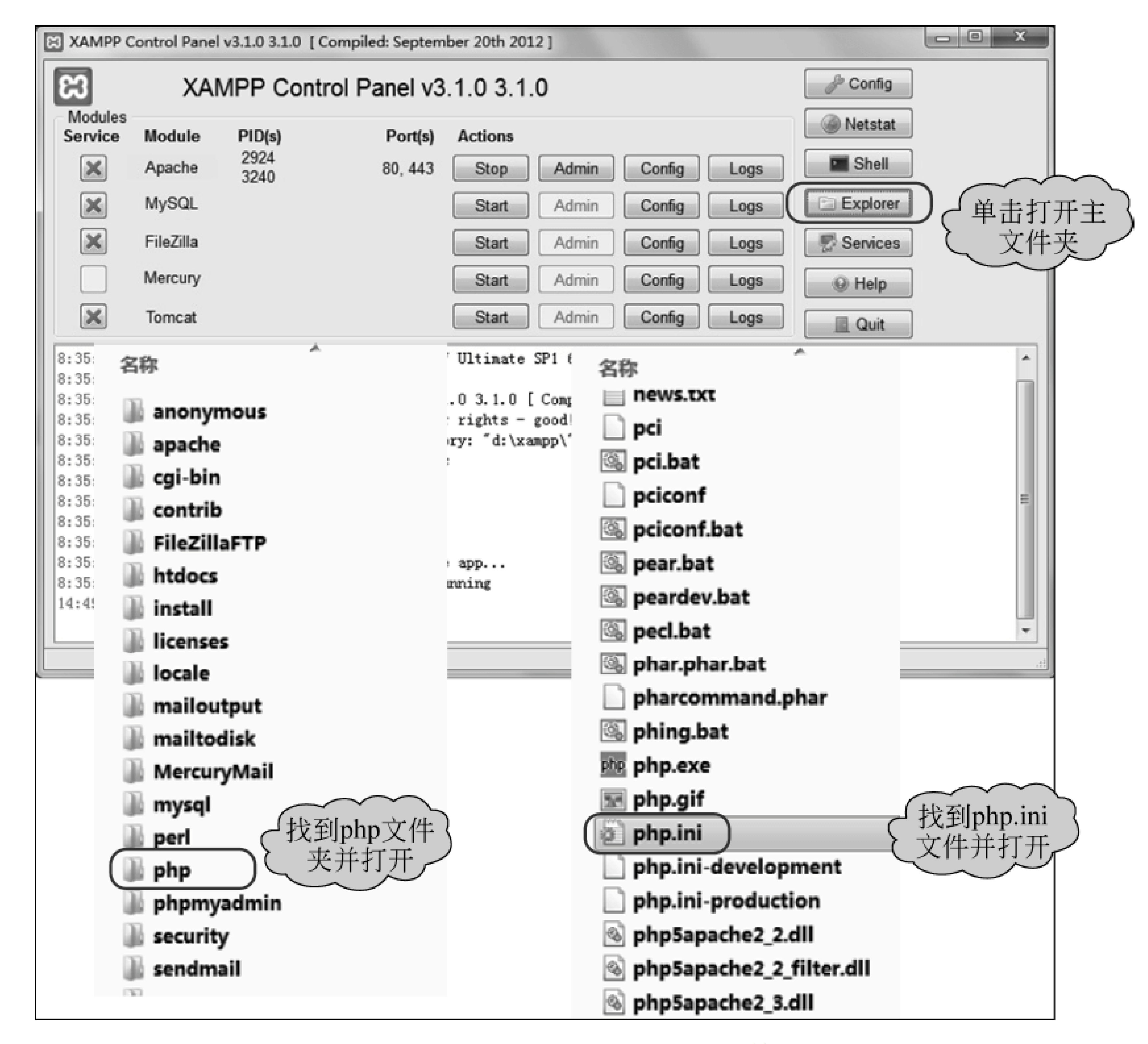


图 7.7　找到 PHP 配置文件

打开后的界面如图 7.8 所示。

```
[PHP]

;;;;;;;;;;;;;;;;;;;
; About php.ini   ;
;;;;;;;;;;;;;;;;;;;
; PHP's initialization file, generally called php.ini, is responsible for
; configuring many of the aspects of PHP's behavior.

; PHP attempts to find and load this configuration from a number of locations.
; The following is a summary of its search order:
; 1. SAPI module specific location.
; 2. The PHPRC environment variable. (As of PHP 5.2.0)
; 3. A number of predefined registry keys on Windows (As of PHP 5.2.0)
; 4. Current working directory (except CLI)
; 5. The web server's directory (for SAPI modules), or directory of PHP
; (otherwise in Windows)
; 6. The directory from the --with-config-file-path compile time option, or the
; Windows directory (C:\windows or C:\winnt)
; See the PHP docs for more specific information.
; http://php.net/configuration.file

; The syntax of the file is extremely simple.  Whitespace and Lines
; beginning with a semicolon are silently ignored (as you probably guessed).
; Section headers (e.g. [Foo]) are also silently ignored, even though
; they might mean something in the future.

; Directives following the section heading [PATH=/www/mysite] only
; apply to PHP files in the /www/mysite directory.  Directives
; following the section heading [HOST=www.example.com] only apply to
; PHP files served from www.example.com.  Directives set in these
; special sections cannot be overridden by user-defined INI files or
; at runtime. Currently, [PATH=] and [HOST=] sections only work under
```

图 7.8　用 Notepad++打开后的 php.ini 文件

我们可以看到文件中有该文件的说明以及一些语法的介绍。该版本的错误配置部分在文件的第 481～666 行之间。我们通常需要配置的代码在第 535 行和 552 行，如下所示。

```
535 error_reporting = E_ALL | E_STRICT
```

```
552 display_errors = On
```

从名称我们就可以看出，error_reportint 选项用于设置报告的错误；display_error 选项用于设置是否将错误显示出来。当然，这些都按照常用的设置为我们设置好了，在需要自定义的时候只需对照表 7.1 中的常量或者值进行设置即可。

2. 使用 error_reporting()函数

通过 php.ini 文件可以设置所有程序默认的错误报告，而通过 error_reporting()函数则可以设置当前程序需要报告的错误，使得错误报告更加灵活。error_reporting()函数的原型如下：

```
int error_reporting ([ int $level ] )
```

参数 level 即为需要设置的报告错误等级，这些错误等级可以使用位运算符来组合使用。如果未设置参数则按照 php.ini 文件配置执行。如果设置为其他不为预定义常量的值，则不显示任何错误。该函数的返回值为旧的错误提示级别，或者在 level 参数未给出时返回当前的级别。

【示例 7-6】以下代码演示输出当前的错误报告级别。

```
<?php
    echo '当前的错误报告级别为：'.error_reporting();
?>
```

代码运行结果如图 7.9 所示。

从运行结果可以看出，错误报告级别以值的形式返回。下面我们就以一个简单的示例来演示在程序中自定义错误报告。

【示例 7-7】以下代码演示不使用 error_reporting()函数设置时将 0 作为除数后的运行结果。

```
<?php
    echo '6/0='.(6/0);
?>
```

代码运行结果如图 7.10 所示。



图 7.9　运行结果



图 7.10　运行结果

从运行结果可以看出，PHP 系统给出了一条 0 作为除数的警告。下面我们就来关闭错误报告，代码如下：

```
<?php
    error_reporting(0);
    echo '6/0='.(6/0);
?>
```

运行后的结果如图 7.11 所示。

从运行结果可以看出，警告信息已经被消除了。当然，我们也可以限定显示某些错误而忽略某些错误。

【示例 7-8】以下代码演示不使用 error_reporting()函数设置时将 0 作为除数，未定义的变量作为被除数后的运行结果。

```
<?php
    echo '$a/0='.$a/0;
?>
```

代码运行结果如图 7.12 所示。



图 7.11　运行结果

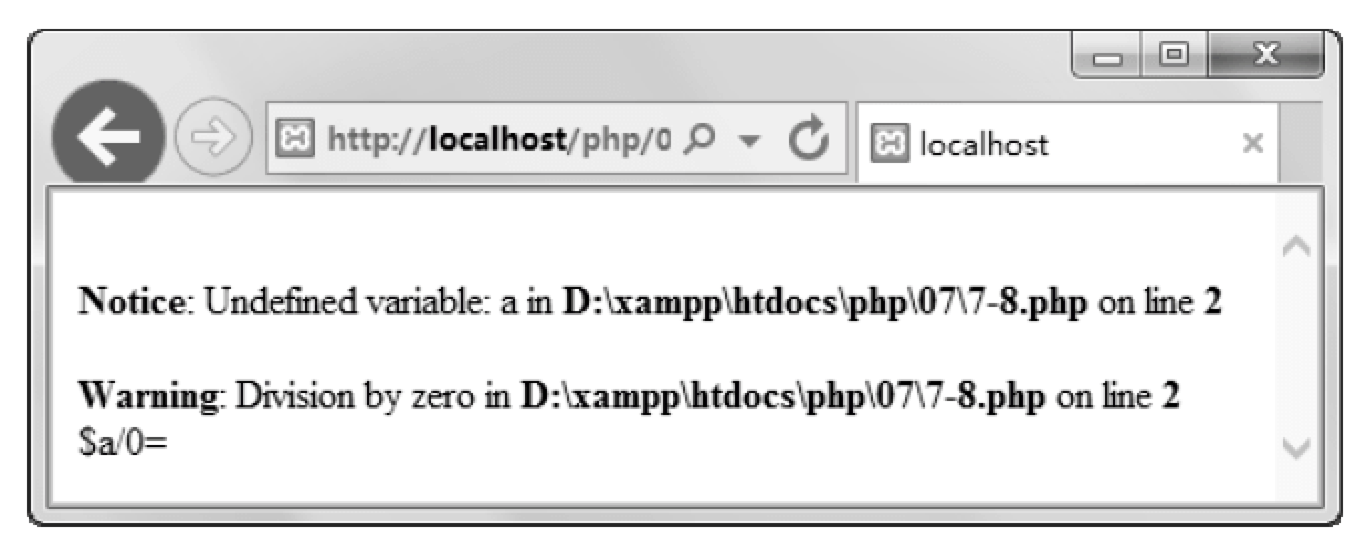


图 7.12　运行结果

从运行结果可以看出，有一条变量为定义的提示和一条 0 作为除数的警告。下面通过设置错误报告来使警告不再提示，代码如下：

```
<?php
    error_reporting(E_ALL&~E_WARNING);
    echo '$a/0='.$a/0;
?>
```

代码运行结果如图 7.13 所示。

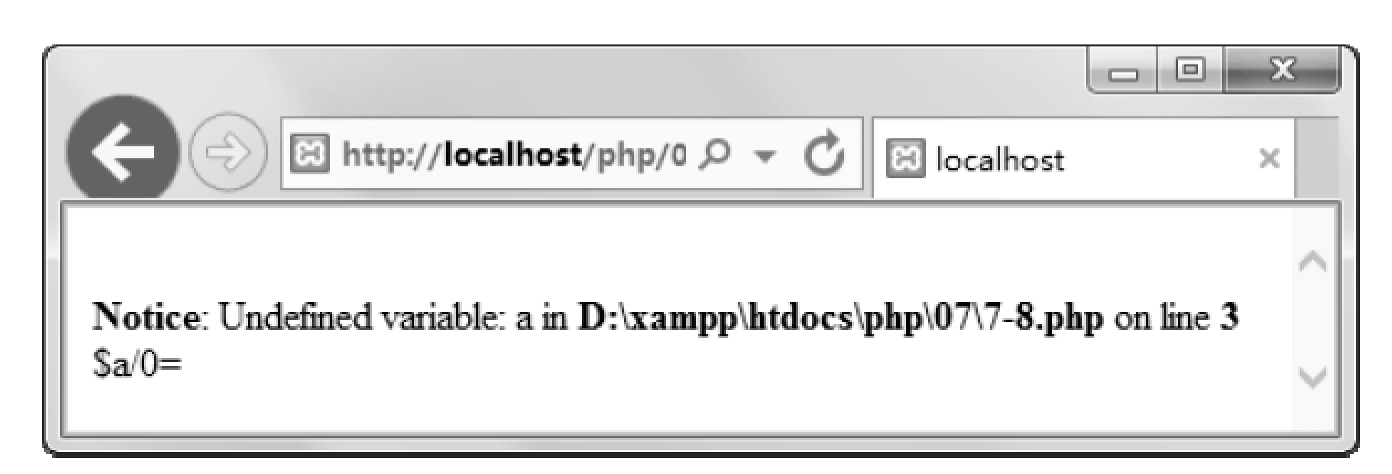


图 7.13　运行结果

从运行结果可以看到，警告信息已经不再显示，而提示信息还在显示，这就演示了使用 error_reporting()函数可以使得 PHP 的错误报告非常灵活。

7.2.3　错误处理

错误处理是代码编写的一个重要部分，前面已经介绍了错误的等级及控制错误提示等级，这些内容都是基于 PHP 内置的错误处理建立的。在本节中将会介绍由我们来自定义错误信息及错误处理函数。

1. die()函数

die()函数是最简单的错误处理函数，它会直接终止当前程序的执行并返回错误信息或

者状态码。die()函数有两种使用形式，函数原型如下：

```
void die ([ string $status ] )
void die ( int $status )
```

第 1 种形式中可选的参数 status 为程序中断后输出的状态信息；第 2 种形式中 status 为 0～254 之间的整数，用来表示程序执行的状态，并且这些状态码不会被输出。状态码 0 表示成功中止程序。

【示例 7-9】以下代码演示使用 die()函数终止程序执行并返回状态信息。

```
01  <?php
02      if(!isset($a)){                 //判断$a 是否定义
03          die('$a 未定义。');          //未定义则终止脚本执行并输出错误信息
04      }
05      echo '程序执行完毕。';           //如果 die()函数被执行则本行代码不会被执行
06  ?>
```

代码运行结果如图 7.14 所示。

从运行结果可以看出，由于代码中的$a 变量未定义，因此 die()函数会立即终止执行程序，所以代码中的第 5 行的输出并不会被执行。事实上，以上代码还有一种简化的方式，就是使用逻辑或（||）来代替 if 判断。

【示例 7-10】以下代码演示使用逻辑或来代替示例 7-9 中的 if 判断来简化代码。

```
<?php
    isset($a)||die('$a 未定义');
    echo '程序执行完毕。';
?>
```

代码运行结果如图 7.15 所示。

我们可以看到该示例的运行结果与示例 7-9 的运行结果一致，但是代码却要简洁一些，这也是 PHP 中常用的一个技巧。其中的原理也很好理解，如果 isset()函数返回值为 TRUE，由于逻辑或的短路原则，则 die()函数不会被执行；否则，die()函数执行。我们也可以通过使 isset()函数返回 TURE 来验证。

【示例 7-11】一些代码演示使 isset()函数返回 TRUE 来使程序执行完毕。

```
<?php
    $a=10;                          //在判断之前定义$a
    isset($a)||die('$a 未定义');
    echo '程序执行完毕。';
?>
```

代码运行结果如图 7.16 所示。



图 7.14　运行结果



图 7.15　运行结果



图 7.16　运行结果

由于$a 已经在判断之前被定义，因此 isset()函数会返回 TRUE，则 die()函数不会被执

行，程序执行完毕后退出。die()函数的参数为整型数的形式比较简单，我来做简单演示即可。

【示例 7-12】以下代码演示使用 die()函数返回状态码。

```
<?php
    isset($a)||die(1);
    echo '程序执行完毕。';
?>
```

代码运行结果如图 7.17 所示。



图 7.17　运行结果

从运行结果可以看出，虽然代码中的$a 并未定义，那么就会导致 die()函数执行，但是运行结果并没有输出 1，原因就在于该参数是作为状态码返回的。

2. 触发错误函数

错误触发函数用来触发一个错误，与 die()函数不同的是，它可以自定义错误信息及错误类型。它的函数原型如下：

```
bool trigger_error ( string $error_msg [, int $error_type = E_USER_NOTICE ] )
```

参数 error_msg 为错误发生时输出的错误信息；可选参数 error_type 为错误的类型，可选的参数如下。

- E_USER_ERROR：致命的用户生成的运行时错误。错误无法恢复，脚本执行被中断。
- E_USER_WARNING：非致命的用户生成的运行时警告。脚本执行不被中断。
- E_USER_NOTICE：默认值。用户生成的运行时通知，脚本发现了可能的错误，也有可能在脚本运行正常时发生。

该函数与 die()函数的不同之处在于，在遇到错误时可以定义要报告的错误级别，而且根据级别的不同可以决定是否继续执行脚本，同时该函数所报告的错误会被 PHP 错误日志记录（如果设置的话，该知识点将在随后的小节中讲解）。

【示例 7-13】以下代码演示使用错误触发函数根据输出不同的值，触发不同的错误。

```
01  <?php
02      $a=9999;
03      //判断$a 的值并触发不同的错误
04      if($a<10){
05          trigger_error('$a<10，计算结果为浮点数');
06      }elseif($a>=10&&$a<10000){
07          trigger_error('10≤$a<10000，计算结果会很小',E_USER_WARNING);
08      }else{
09          trigger_error('给定的除数太大！',E_USER_ERROR);
10      }
11      echo "10/{$a}=",10/$a;
12      echo '<br />程序执行完毕。';
13  ?>
```

代码运行结果如图 7.18 所示。

从运行结果可以看出，虽然代码触发了 Notice 级别的错误，程序仍然会执行完毕。我们将代码第 2 行中的$a 变量赋值为一个大于等于 10 并且小于 10000 的值后（这里设置为

9999)，运行结果如图 7.19 所示。



图 7.18　运行结果



图 7.19　运行结果

从运行结果可以看出，代码触发了一个 Warning 级别的错误。接下来将代码第 2 行中的$a 变量的值赋值为一个大于或者等于 10000 的值后（这里为 10001），运行结果如图 7.20 所示。



图 7.20　运行结果

我们可以看到，代码触发了一个 Fatal 级别的错误，而且导致程序终止运行，程序没有输出结果。

3. 自定义错误处理器

前面所介绍的内容中，虽然我们可以自己触发一个错误并且可以定义错误信息和类型，但是这些错误都是由系统处理器来处理的。下面就来介绍自定义错误处理器的方法：

（1）首先需要定义一个错误处理的函数，该函数至少需要接受两个参数，完整的形式如下：

```
函数名(error_level,error_message[,error_file,error_line,error_context]){
    //语句
}
```

该函数的参数描述如表 7.2 所示。

（2）在定义好一个我们自己的错误处理函数后，它还不能用来处理错误，还需要使用 set_error_handler()函数来将其设置为错误处理器，该函数的原型如下：

```
mixed set_error_handler ( callable $error_handler [, int $error_types = E_ALL
| E_STRICT ] )
```

表 7.2　自定义错误处理函数参数描述

参　数	描　述
error_level	必需。为用户定义的错误规定错误报告级别
error_message	必需。为用户定义的错误规定错误消息
error_file	可选。规定错误在其中发生的文件名
error_line	可选。规定错误发生的行号
error_context	可选。规定一个数组，包含了当错误发生时在用的每个变量及它们的值

参数 error_handler 即为自定义的错误处理函数；可选参数 error_types 则为该函数要处理的错误类型。

下面就来定义一个简单错误处理函数，如下所示。

```
function myerror($errno,$errmsg){
    echo "这里发生了错误！错误号为：[{$errno}]，错误信息是：{$errmsg}.";
    return TRUE;
}
```

该函数的返回值用于控制是否同时执行 PHP 内置的错误处理器。如返回 TRUE 则不执行；如返回 FALSE 则执行。

【示例 7-14】以下代码演示使用自定义的错误处理器来处理错误。

```
<?php
    function myerror($errno,$errmsg){   //定义错误处理函数
        echo "这里发生了错误！错误号为：[{$errno}]，错误信息是：{$errmsg}.";
        return TRUE;
    }
    set_error_handler('myerror');        //设置错误处理器
    echo $a;                             //输出未定义的变量，会触发错误
?>
```

代码运行结果如图 7.21 所示。



图 7.21　运行结果

从运行结果可以看出，当错误被触发的时候，调用了我们自定义的错误处理器来处理。我们还可以通过设置自定义函数的返回值来同时执行系统内置的错误处理器。

【示例 7-15】以下代码演示使用自定义错误处理器的同时让系统内置错误处理器同时运行。

```
<?php
    function myerror($errno,$errmsg){   //定义错误处理函数
        echo "这里发生了错误！错误号为：[{$errno}]，错误信息是：{$errmsg}.";
        return FALSE;          //返回 FALSE，让系统内置错误处理函数同时执行
    }
    set_error_handler('myerror');        //设置错误处理器
```

```
    echo $a;                                //输出未定义的变量，会触发错误
?>
```

代码运行结果如图 7.22 所示。



图 7.22　运行结果

从以上运行结果可以看出，程序在执行我们自定义错误处理器的同时，让系统内置错误处理器也在同时执行。当然，自定义错误处理器的作用远不是这么简单，一个实用的情况，就是判断出现的错误类型，例如出现比较严重的错误时，可以发送邮件给维护人员；而出现不严重的错误时，可以将错误信息记录在文件中。

【示例 7-16】以下代码演示一个自定义错误处理器的实用案例。

```
01  <?php
02      function myerror($errno,$errmsg){       //定义错误处理函数
03          if($errno==E_USER_NOTICE||$errno==E_USER_WARNING){
04              error_log("发生了不严重的错误，错误号为{$errno}，错误信息为：
                    {$errmsg}。");
05          }else{
06              error_log("发生了严重的错误！错误信息为：{$errmsg}。",
                    1,'myemail@email.com');
07          }
08          return TRUE;
09      }
10      function testerror($a){                 //定义含有错误触发语句的函数
11          if($a<10){
12              trigger_error('$a<10，计算结果为浮点数');
13          }elseif($a>=10&&$a<10000){
14              trigger_error('10≤$a<10000，计算结果会很小',E_USER_WARNING);
15          }else{
16              trigger_error('给定的除数太大！',E_USER_ERROR);
17          }
18          return 10/$a;
19      }
20      set_error_handler('myerror');           //设置错误处理器
21      echo testerror(999);                    //调用函数
22  ?>
```

注意：代码中的 error_log()函数用于向指定的地方发送信息。

代码运行结果如图 7.23 所示。

从运行结果可以看到，程序在第 21 行调用触发错误的函数后输出了计算结果，而错误信息并没有被显示出来。下面就来看一下 PHP 默认的日志，如图 7.24 所示。

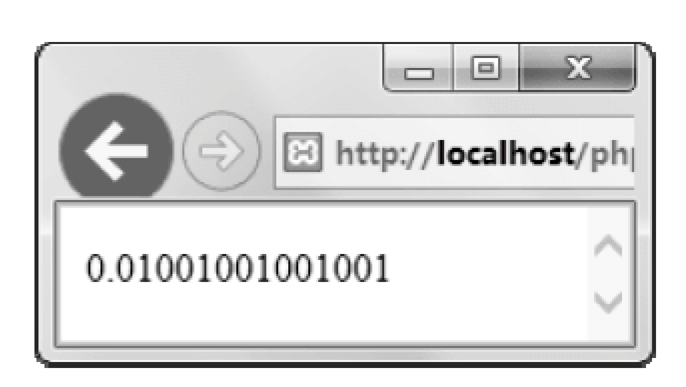


图 7.23　运行结果

我们可以看到，在 PHP 日志中被写入了错误信息。接下来就介绍错误记录的知识。

```
php_error_log - 记事本
文件(F)  编辑(E)  格式(O)  查看(V)  帮助(H)
[07-Jan-2013 08:12:14 UTC] 发生了不严重的错误，错误号为512，错误信息为：10<$a<10000，计算
结果会很小。
```

图 7.24　PHP 错误记录文件

4. 记录错误到文件

在我们自己创建的网站投入运营之后，通常就会将错误提示关闭，这是因为用户并不希望看到这些错误代码。而错误的信息将会记录在文件中以供维护系统之用。错误记录的配置也是由 php.ini 文件设置的。该版本所在的位置及设置如下：

```
574 log_errors = On
579 log_errors_max_len = 1024
659 error_log =  "D:\xampp\php\logs\php_error_log"
```

从配置的名称我们也可以很容易地理解：log_error 用于设置错误记录打开或者关闭；log_errors_max_len 用于设置错误记录的大小，以字节为单位；error_log 用于设置错误记录的文件。我们就可以根据配置文件所设置的目录打开错误记录，如图 7.25 所示。

```
php_error_log - 记事本
文件(F)  编辑(E)  格式(O)  查看(V)  帮助(H)
\php\5\5-14.php on line 7
[11-Dec-2012 06:22:51 UTC] PHP Notice:  Undefined offset: 2160 in D:\xampp\htdocs
\php\5\5-14.php on line 9
[11-Dec-2012 06:22:51 UTC] PHP Notice:  Undefined offset: 2161 in D:\xampp\htdocs
\php\5\5-14.php on line 7
[11-Dec-2012 06:22:51 UTC] PHP Notice:  Undefined offset: 2161 in D:\xampp\htdocs
\php\5\5-14.php on line 9
[11-Dec-2012 06:22:51 UTC] PHP Notice:  Undefined offset: 2162 in D:\xampp\htdocs
\php\5\5-14.php on line 7
[11-Dec-2012 06:22:51 UTC] PHP Notice:  Undefined offset: 2162 in D:\xampp\htdocs
\php\5\5-14.php on line 9
[11-Dec-2012 06:22:51 UTC] PHP Notice:  Undefined offset: 2163 in D:\xampp\htdocs
\php\5\5-14.php on line 7
[11-Dec-2012 06:22:51 UTC] PHP Notice:  Undefined offset: 2163 in D:\xampp\htdocs
\php\5\5-14.php on line 9
[11-Dec-2012 06:22:51 UTC] PHP Notice:  Undefined offset: 2164 in D:\xampp\htdocs
\php\5\5-14.php on line 7
[11-Dec-2012 06:22:51 UTC] PHP Notice:  Undefined offset: 2164 in D:\xampp\htdocs
\php\5\5-14.php on line 9
[11-Dec-2012 06:22:51 UTC] PHP Notice:  Undefined offset: 2165 in D:\xampp\htdocs
\php\5\5-14.php on line 7
[11-Dec-2012 06:22:51 UTC] PHP Notice:  Undefined offset: 2165 in D:\xampp\htdocs
\php\5\5-14.php on line 9
[11-Dec-2012 06:22:51 UTC] PHP Notice:  Undefined offset: 2166 in D:\xampp\htdocs
\php\5\5-14.php on line 7
[11-Dec-2012 06:22:51 UTC] PHP Notice:  Undefined offset: 2166 in D:\xampp\htdocs
```

图 7.25　错误记录文件

该文件记录了自服务器运行以来的所有出现的错误。下面我们就来演示向日志文件中写入一条记录。error_log()函数可以用来发送信息到指定的位置，原型如下：

```
bool error_log ( string $message [, int $message_type = 0 [, string
$destination [, string $extra_headers ]]] )
```

必须的参数 message 为要发送的错误信息；参数 message 为发送的信息类型，可选的参数如下。

- ❑ 0：message 发送到 PHP 的系统日志，使用操作系统的日志机制或者一个文件，取决于 error_log 指令设置了什么。这是个默认的选项。
- ❑ 1：message 发送到参数 destination 设置的邮件地址。第 4 个参数 extra_headers 只

有在这个类型里才会被用到。

- 3：被发送到位置为 destination 的文件里。字符 message 不会作为新行写入。
- 4：message 直接发送到 SAPI 的日志处理程序中。

为了使结果更加容易查看，这里我们将原来的错误日志文件删除后再进行示例的演示。

【示例 7-17】以下代码演示向日志文件中写入一条记录。

```
01  <?php
02      $y=0;
03      if($y==0){
04          error_log('0 被作为除数，无法运行计算。'); //将信息写入到配置的文件中
05      }
06      echo '程序运行结束。';
07  ?>
```

代码运行结果如图 7.26 所示。

从运行结果可以看出程序已经执行完毕。由于第 2 行代码的变量值为 0，所以 if 判断的条件成立，因此 error_log()函数被执行。我们打开日志文件就可以看到写入日志中的信息。如图 7.27 所示。



图 7.26　运行结果



图 7.27　日志文件

当然，我们也可以将错误信息发送到指定的邮箱中。邮件发送成功后我们将会收到一个标题为 PHP error_log message 的邮件，这里不再详细演示。

5. 记录错误到系统日志

除了可以将日志记录到文件中之外，还可以将这些日志记录到操作系统日志中，这样的优势在于可以更好地管理和查看日志。error_log()函数也可以用于向操作系统发送日志消息，但这取决于 PHP 的配置。我们只要将 php.ini 做如下修改即可：

```
error_log = syslog
```

修改以上配置后，error_log()函数就会将日志写入到操作系统日志中。需要注意的是，Windows 操作系统目前只支持 gb2312 编码格式的中文字符。如果日志消息中使用其他编码格式的中文字符，则有可能会被日志系统忽略。

【示例 7-18】以下代码演示使用 error_log()函数将日志写入操作系统日志。

```
<?php
    $y=0;
    if($y==0){
        error_log('0 被作为除数，无法运行计算。');//将信息写入到配置的文件中
    }
    echo '程序运行结束。';
?>
```

代码运行结果如图 7.28 所示。

我们可以看到该示例的代码与上一个示例完全相同，区别就在于我们将配置文件改为了向操作系统中写入日志消息。在 Windows 操作系统中的日志可以使用事件查看器查看。在开始菜单中搜索“事件查看器”程序并打开，然后在左侧的导航窗口中选择“Windows 日志”中的“应用程序”项即可，如图 7.29 所示。



图 7.28　运行结果



图 7.29　Windows 事件查看器

图中的最新消息记录即为我们使用上面的示例代码写入的，可以通过查看事件属性来获取接收到的消息。该事件的详细信息如图 7.30 所示。

以上信息提供了详细的内容供我们查看，但是通常情况下并不使用 error_log ()函数向操作系统日志发送消息。通常会使用 openlog()、closelog()和 syslog()函数向操作系统中发送消息，使用这 3 个函数的优势就在于可以更详细地定制这些日志消息。这 3 个函数的使用方法比较简单，我们首先来看这 3 个函数的原型：

```
bool openlog ( string $ident , int $option , int $facility )
bool closelog ( void )
bool syslog ( int $priority , string $message )
```

open()函数用于打开至系统日志记录的连接，该函数在记录日志的过程中是可选的。参数 ident 用于在每条信息前加入一个字符串；参数 option 用于指示日志生成时使用哪个选择，可以选择的参数如下。

- LOG_CONS：写入 syslog 时发生错误，则将输出发送到系统控制台。
- LOG_NDELAY：立即打开与 syslog 的连接。
- LOG_NDELAY：提交第一条信息后打开与 syslog 的连接，这是默认值。



图 7.30　事件的详细信息

- LOG_PERROR：记录的消息同时输出到 syslog 和标准错误。
- LOG_PID：每个消息都带有进程 ID。

注意：LOG_PID 选项是 Windows 操作系统中唯一支持的选项。

参数 facility 用于指定日志消息的类型，可以选择的参数如下。

- LOG_AUTH：安全的/授权的信息（代替系统中默认的定义）。
- LOG_AUTHPRIV：安全的/授权的信息（私有的）。
- LOG_CRON：时钟进程。
- LOG_DAEMON：其他系统进程。
- LOG_KERN：内核信息。
- LOG_LOCAL0 ... LOG_LOCAL7：本地保留。
- LOG_LPR：行输出子系统。
- LOG_MAIL：电子邮件子系统。
- LOG_NEWS：USENET 消息子系统。
- LOG_SYSLOG：系统日志。
- LOG_USER：一般用户日志。
- LOG_UUCP：UUCP 子系统。

注意：LOG_USER 选项是 Windows 操作系统中唯一支持的选项。

Closelog()函数用于关闭与日志记录的连接，该函数在记录日志的过程中是可选的。

Syslog()函数用于生成一个系统日志，在 3 个函数中是最主要的。参数 priority 用来表示日志的等级，可以选择的参数如下。

- LOG_EMERG：系统崩溃。
- LOG_ALERT：必须处理的活动。
- LOG_CRIT：危险的状态。
- LOG_ERR：错误信息。
- LOG_WARNING：警告信息。
- LOG_NOTICE：提醒信息。
- LOG_INFO：报告信息。
- LOG_DEBUG：调试信息。

参数 message 为日志的内容。

【示例 7-19】以下代码演示使用 syslog()函数向系统中写入日志。

```
<?php
    $y=0;
    if($y==0){
        syslog(LOG_WARNING,'0 被作为除数，无法运行计算。');
                                                //将信息写入到配置的文件中
    }
    echo '程序运行结束。';
?>
```

代码运行结果如图 7.31 所示。接下来就可以查看系统日志，最新的日志消息如图 7.32 所示。



图 7.31　运行结果



图 7.32　事件详细信息

而与 error_log()函数不同的是，我们使用 syslog()函数定义了日志的级别，如图 7.33 所示。



图 7.33　日志的级别

虽然在 Windows 操作系统中 openlog()函数可以使用的选项比较少，但是我们仍然可以使用它来为每条信息加入一个字符串。

【示例 7-20】以下代码演示使用 openlog()函数为日志信息增加新字符串。

```
<?php
    $y=0;
    if($y==0){
        openlog('我的 PHP 程序',LOG_PID,LOG_USER);
        syslog(LOG_WARNING,'0 被作为除数，无法运行计算。');
                                                    //将信息写入到配置的文件中
    }
    echo '程序运行结束。';
?>
```

代码运行结果如图 7.34 所示。最新的日志消息如图 7.35 所示。



图 7.34　运行结果

需要注意的是，syslog()函数向系统写入日志并不会受到 PHP 配置文件的影响。

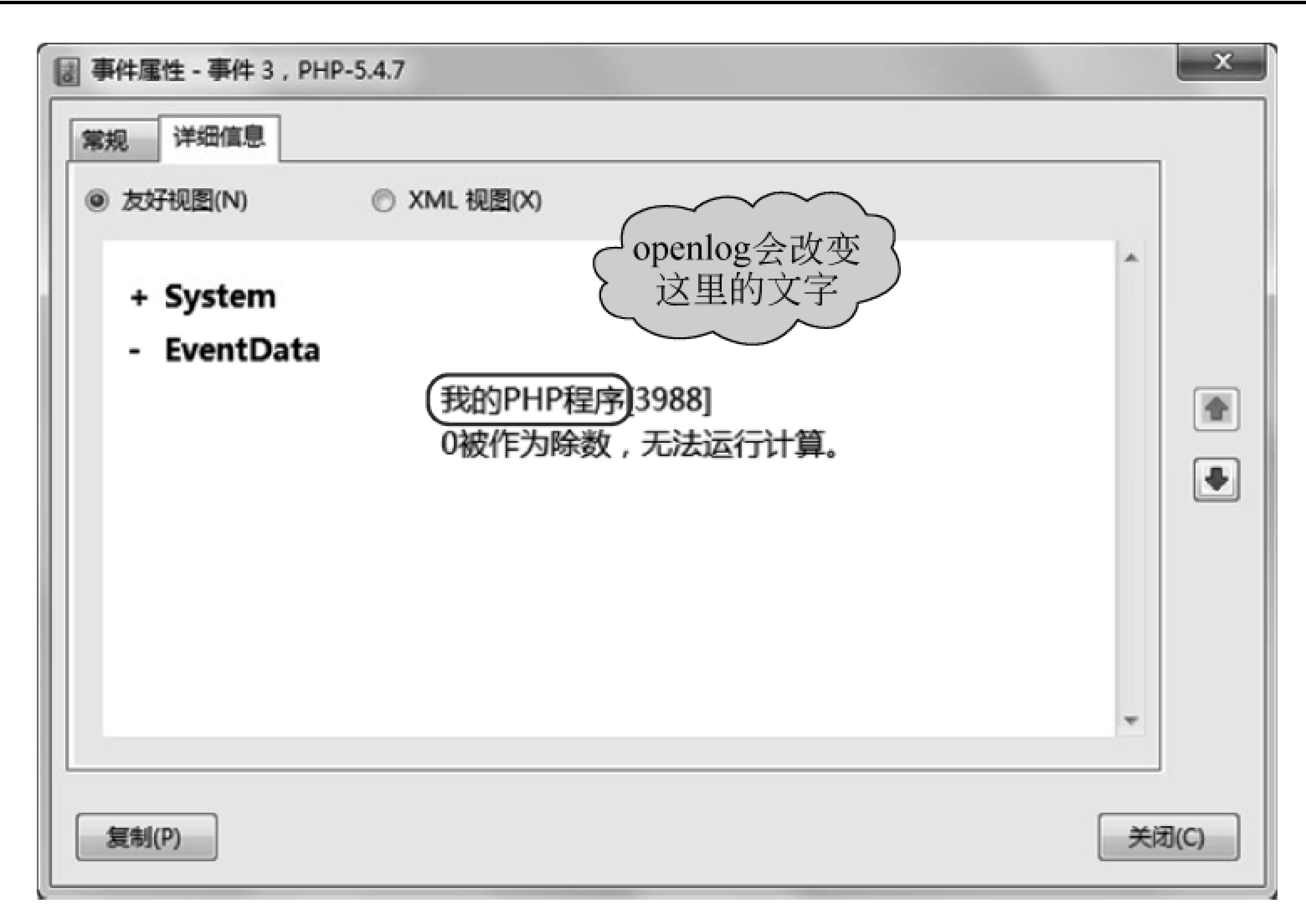


图 7.35　日志详细信息

7.2.4　异常

PHP 标准错误虽然可以向我们报告一些信息，但它也是有局限性的，例如它的错误信息不够明确，以及错误必须马上被处理，而且错误的出现通常会立即终止程序执行并且错误不可以在代码中被处理。因此在 PHP5 中引入了异常这种新的错误处理机制，来弥补标准错误处理的不足。

在 PHP 中的异常表示程序在执行过程中，发生了一个中断程序正常执行的异常事件，异常并不是指程序语法出现错误。在有可能出现异常的地方就可以设置报告异常的语句。异常使用 throw 语句来报告。报告异常需要通过实例化一个 PHP 内置异常类 Exception 或者其子类的对象。

```
throw new Exception 或者其子类(异常信息);
```

【示例 7-21】以下代码演示报告一个异常。

```
<?php
    if(!isset($i)){                                //判断变量是否定义
        throw new Exception('$i 未定义');          //报告一个异常
    }
    echo '程序运行结束。';
?>
```

代码运行结果如图 7.36 所示。



图 7.36　运行结果

以上代码就报告了一个异常信息，由于我们并没有设置处理这个异常的代码，因此代

码会报出一个未捕获异常的错误。

7.2.5　处理异常

PHP 中的异常是通过捕获一个内置异常类 Exception 类或者其子类的对象来处理的。Exception 类是 PHP 所有异常类的父类，它的结构如下：

```
Exception {
    /* 属性 */
    protected string $message;           //异常消息
    protected int $code;                 //异常代码
    protected string $file;              //出现异常的文件
    protected int $line;                 //抛出异常的代码所在行
    /* 方法 */
    public __construct ([ string $message = "" [, int $code = 0 [, Exception
$previous = NULL ]]] )
    final public string getMessage (void)
    final public Exception getPrevious (void)
    final public int getCode (void)
    final public string getFile (void)
    final public int getLine (void)
    final public array getTrace (void)
    final public string getTraceAsString (void)
    public string __toString (void)
    final private void __clone (void)
}
```

以上加粗的代码为可以在其子类中进行扩展的方法。

1. 捕获并处理异常

捕获一个异常需要使用 try…catch 结构，它的形式如下：

```
try{
    //可能发生异常的代码
}catch(异常类的对象){
    //异常处理代码
}
```

【示例 7-22】以下代码演示捕获并处理一个异常。

```
<?php
    try{
        if(!isset($i)){                             //判断变量是否定义
            throw new Exception('$i 未定义');       //报告一个异常
        }
        echo '程序运行结束。';                      //该代码不会被执行
    }catch(Exception $e){                           //捕获并处理异常
        echo '程序未正常执行完毕！'.'错误信息为：'.$e->getMessage();
    }
?>
```

代码运行结果如图 7.37 所示。

这里我们只演示了一个最简单的异常抛出与捕获并处理的过程，在实际应用中处理的代码可以做得比较实用。

2. 扩展内置异常处理类

异常处理类可以通过继承来实现扩展，以用来满足不同的使用环境。

【示例 7-23】以下代码演示扩展内置异常处理类来增加功能。

```
<?php
    class myException extends Exception{//自定义异常类，继承自 Exception 类
        protected $now;              //自定义的变量
        public function __construct($message=NULL,$code=0){
                                     //覆盖父类的构造方法
            $this->now=time();       //获取当前时间
            parent::__construct($message,$code);
                                     //调用父类的构造方法以保证数据正确初始化
        }
        public function gettime(){                    //自定义的方法
            return date('Y-m-d',$this->now);          //格式化后输出时间
        }
    }
    try{
        throw new myException();                      //报告一个异常
    }catch(myException $e){
        echo '异常发生的时间为：'.$e->gettime();       //捕获异常并处理
    }
?>
```

代码运行结果如图 7.38 所示。



图 7.37　运行结果



图 7.38　运行结果

以上的代码就为我们自定义的异常类增加了获取异常发生时间的功能。

3. 捕获并处理多个异常

捕获并处理多个异常使用的结构如下：

```
try{
    //可能发生异常的代码
}catch(异常类的对象){
    //异常处理代码
}catch(异常类的对象){
    //异常处理代码
}…
```

捕获多个异常就需要代码会抛出多个异常，前面我们已经学习了通过扩展内置的异常处理类，来增加异常处理的功能。那么就可以抛出多个异常类和其子类的对象来报告异常。

【示例 7-24】以下代码演示捕获并处理多个异常。

```
<?php
    class myException extends Exception{//自定义异常类，继承自 Exception 类
```

```
03          protected $now;                   //自定义的变量
04          public function  __construct($message=NULL,$code=0){
                                              //覆盖父类的构造方法
05              $this->now=time();
06              parent::__construct($message,$code);
                                      //调用父类的构造方法以保证数据正确初始化
07          }
08          public function gettime(){        //自定义的方法
09              return date('Y-m-d',$this->now);
10          }
11      }
12      class writetolog extends Exception{//自定义异常类，继承自 Exception 类
13          public function writetolog(){
14              error_log('未知错误！');       //将错误信息写入日志
15          }
16      }
17      $x=5;                                 //定义一个变量
18      try{
19          if($x>0){
20              throw new myException();      //报告一个 myException 类型的异常
21          }elseif($x==0){
22              throw new Exception('除数为 0');      //报告一个标准异常
23          }else{
24              throw new write_log();        //报告一个 writetolog 类型的异常
25          }
26      }catch(myException $e){
27          echo '异常发生的时间为：'.$e->gettime();
                                              //捕获 myException 类型异常并处理
28      }catch(writetolog $e){
29          echo '已经将异常信息写入日志。';   //捕获 writetolog 类型异常并处理
30          $e->write_log();
31      }catch(Exception $e){
32          echo '错误信息为：'.$e->getMessage();     //捕获内置异常类并处理
33      }
34  ?>
```

代码运行结果如图 7.39 所示。当将第 17 行中的变量 x 的值改为 0 后，运行结果如图 7.40 所示。当将第 17 行中的变量 x 的值改为一个小于 0 的数值后，运行结果如图 7.41 所示。




图 7.39　运行结果



图 7.40　运行结果



图 7.41　运行结果

以上运行结果提示异常信息已经写入日志，我们可以打开日志文件来查看，如图 7.42 所示。

以上就完整地演示了捕获并处理多个异常的情况，在实际应用中读者可以根据情况增减自定义异常处理类。

4. 自定义异常处理器

同自定义错误处理器类似，我们也可以来自定义异常处理器。使用的函数为 set_exception_handler()，它的原型如下：

```
callable set_exception_handler ( callable $exception_handler )
```

参数 exception_handler 为一个回调函数，该函数应该支持一个参数用来接收未捕获异常类的对象。

【示例 7-25】以下代码演示设置一个自定义异常处理器来处理未捕获的异常。

```
<?php
    function myhandler($e){                    //定义异常处理函数
        echo '有一个异常未被捕获！错误信息为：'.$e->getMessage();
        echo '<br />所在的文件为：'.$e->getFile();
    }
    set_exception_handler('myhandler');        //设置异常处理函数
    if(!isset($i)){                            //判断变量是否定义
        throw new Exception('$i 未定义');      //报告一个异常
    }
    echo '程序运行结束。';
?>
```

代码运行结果如图 7.43 所示。



图 7.42　日志文件



图 7.43　运行结果

该函数的使用比较简单，这里就不再多做介绍。

5. 再次抛出异常

在捕获到一个异常以后，会遇到无法处理的情况，那么就可以将这个异常再次报告，直到被处理或者被系统错误处理器捕获。

【示例 7-26】以下代码演示在异常处理结构中再次抛出异常。

```
01  <?php
02      $x=5;
03      try{
04          if($x>0){
05              throw new Exception('$x>0');                //报告一个异常
06          }else{
07              throw new Exception('unknow Exception');    //报告一个异常
08          }
09      }catch(Exception $e){
10          if($e->getMessage()=='unknow Exception'){
11              throw new Exception('无法处理的异常');       //再次报告异常
12          }
13          echo '捕获到异常，异常信息为：'.$e->getMessage();
14      }
15  ?>
```

以上代码运行结果如图 7.44 所示。当把第 2 行代码中的变量 x 值改为小于或者等于 0 的数值后，运行结果如图 7.45 所示。

从运行结果可以看到，代码发出了一个未捕获异常的错误。由于该异常是在错误处理的过程中再次报告的，因此我们需要再设置高一级的错误处理结构来捕获这个异常。



图 7.44　运行结果

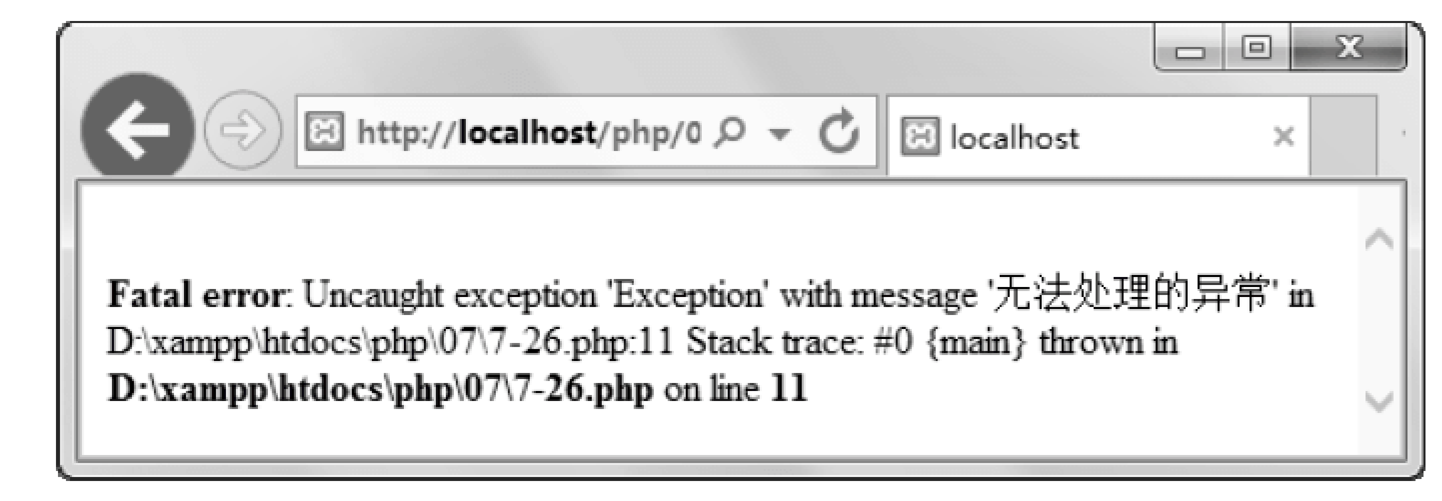


图 7.45　运行结果

【示例 7-27】以下代码演示捕获在异常处理过程中报告的新异常。

```
01  <?php
02      try{
03          $x=5;
04          try{
05              if($x>0){
06                  throw new Exception('$x>0');            //报告一个异常
07              }else{
08                  throw new Exception('unknow Exception');//报告一个异常
09              }
10          }catch(Exception $e){
11              if($e->getMessage()=='unknow Exception'){
12                  throw new Exception('无法处理的异常');    //再次报告异常
13              }
14              echo '一级错误处理结构捕获到异常,异常信息为:'.$e->getMessage();
15          }
16      }catch(Exception $e){
17          echo '二级错误处理结构捕获到异常，异常信息为：'.$e->getMessage();
18      }
19  ?>
```

代码运行结果如图 7.46 所示。我们将第 3 行代码改为一个小于等于 0 的数值后，运行结果如图 7.47 所示。



图 7.46　运行结果



图 7.47　运行结果

以上示例就简单演示了再次报告异常和捕获异常的过程，当然这个错误还可以再次向上报告，这里就不再讲解。

7.3　小　　结

本章主要介绍了 PHP 的错误处理部分的知识。由于现在我们还不具备能处理常见错误的知识，因此多处的知识都是从原理性的方面做了介绍，但是并不会影响到读者的学习。异常处理在实际应用中属于比较高级的部分，不会立即被应用，读者在有了一定的积累后就可以灵活使用错误处理了。

7.4 本章习题

1. 回答错误可能发生的 4 种原因及它们的特点。
2. 回答最常见的 4 种错误等级及它们对程序执行的影响。
3. 判断下列代码执行时会报告哪种错误级别。

第一段代码：

```
<?php
    for($i=0;$i<5;$i++)
        echo $j;
?>
```

第二段代码：

```
<?php
    function myfunc($str1,$str2)
        return $str1.$str2;
    echo myfunc(1,'two');
?>
```

4. 判断以下代码片段属于哪类错误。

```
fucntion myerror(){
    echo "请找出错误";
}
myerror();
```

```
if($a=0){
    echo "0 不能作为除数。";
}
```

第 2 篇　PHP 开发进阶

第 8 章　字符串处理

字符串的输入与输出可以说是 PHP 的精华所在，因而 PHP 中提供了非常丰富的字符串处理函数来应用于各种环境。本章我们就主要介绍这些函数和它们可以使用的场景。

8.1　输出字符串

在输出字符串函数中一直以来最多使用的是 echo()函数，它可以说是我们接触最早的一个函数，也是使用最为频繁的一个函数。除了 echo()函数之外，我们还使用 die()函数，它的主要用途不是为了输出字符串，但是它也可以输出字符串。本节我们将要介绍的是除了这两个我们使用过的函数之外，还有其他的一些输出字符串的函数。

8.1.1　print()函数

print()函数与 echo()函数的作用类似，他们的主要的不同之处在于 print()函数总会返回 1，它的原型如下：

```
int print ( string $arg )
```

参数 arg 即为需要输出的内容；从严格意义上来说，print 为一个语言结构，因此我们可以不使用圆括号来包围它的参数列表。其他的使用方法就都与 echo()函数类似，例如，支持解析双引号字符串中的变量名等特性。

【示例 8-1】以下代码演示 print()函数的用法。

```
<?php
    $str='Hello PHP!';                      //定义一个字符串变量
    print "使用 print 函数输出{$str}";      //使用 print 函数输出字符串
?>
```

代码运行结果如图 8.1 所示。



图 8.1　运行结果

由于 print()函数的使用方法非常简单，因此这里不再详细介绍。

8.1.2　格式化字符串函数

格式化字符串函数用于将字符串按照指定的格式格式化并返回或者输出。PHP 提供了多个格式化字符串的函数，如下所示。

```
int printf (string $format [, mixed $args [, mixed $... ]])
string sprintf (string $format [, mixed $args [, mixed $... ]])
int vprintf (string $format, array $args)
string vsprintf (string $format, array $args)
```

以上 4 个函数中的 format 参数为格式化字符串；参数 args 为参量表，参量表中的值会按照 format 参数中的格式进行格式化。printf()函数和 vprint()函数会将格式化后的字符串输出。sprintf()和 vsprintf()函数会将格式化后的字符串返回而不做输出。在介绍这些函数的使用方法以前，首先我们需要学习格式化字符串的形式。常用的格式化字符如下：

- %d：十进制有符号整数。
- %u：十进制无符号整数。
- %f：浮点数。
- %s：字符串。
- %c：单个字符（ASCII 码值）。
- %e：指数形式的浮点数。
- %x：以十六进制表示的无符号整数（小写字母形式）。
- %X：以十六进制表示的无符号整数（大写字母形式）。
- %o：无符号以八进制表示的整数。

这些格式化字符可以加入到字符串中，让格式化字符串函数进行相应的处理。

【示例 8-2】以下代码演示使用格式化字符串，将浮点数格式化为十进制有符号整数后输出。

```
<?php
    $num=123.456;                          //定义一个浮点数变量
    printf('以整数形式输出：%d',$num);      //格式化为有符号十进制整数后输出
?>
```

代码运行结果如图 8.2 所示。需要注意的是，将浮点数格式化为有符号十进制数会直接将浮点数部分舍去，而不会进行四舍五入。格式化字符串中的格式化字符需要有对应的参数与它匹配，否则就会出现一个参数太少的警告信息。

【示例 8-3】以下代码演示正确使用格式化字符串函数的形式。

```
01  <?php
02      $num=123456.789;          //定义一个浮点数
03      //以两种形式格式化输出
04      printf('以整数形式输出：%d，以指数形式输出：%e',$num,$num);
05  ?>
```

代码运行结果如图 8.3 所示。

第 4 行代码中有两个格式化字符“%d”和“%e”，那么我们就需要两个参数与其对应。但是我们可以看到这两个格式化字符都对应于一个参数，那么就可以使用占位符来省略多



图 8.2　运行结果

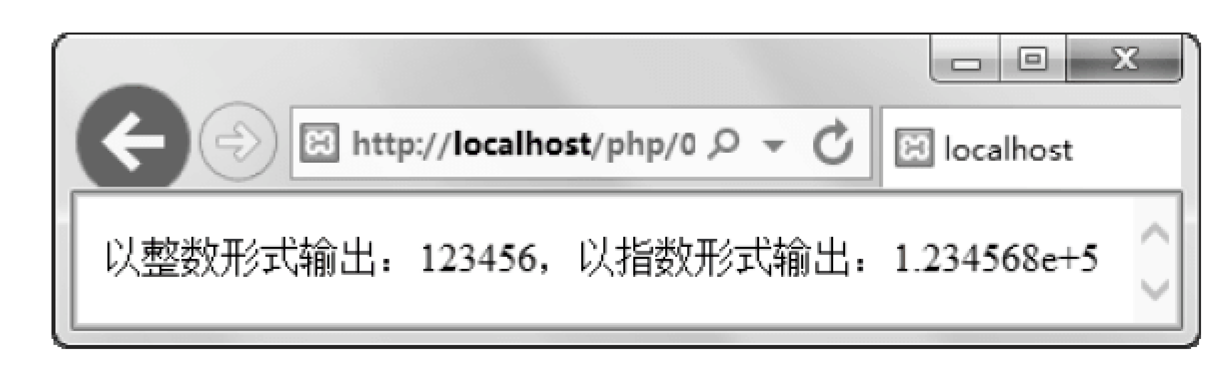


图 8.3　运行结果

次传入相同的参数。占位符被插入在%符号之后，由数字和“$”组成。使用的形式如下：

```
printf('以整数形式输出：%d，以指数形式输出：%e',$num,$num)
printf('以整数形式输出：%1$d，以指数形式输出：%1$e',$num)
```

注意：在双引号形式的字符串中，需要将“$”符号转义，即为“\$”形式。

【示例 8-4】以下代码演示使用占位符修改示例 8-3 中的代码。

```
<?php
    $num=123456.789;          //定义一个浮点数
    // 以两种形式格式化输出
    printf('以整数形式输出：%1$d，以指数形式输出：%1$e',$num);
?>
```

代码运行结果如图 8.4 所示。



图 8.4　运行结果

从运行结果可以看出，代码在改写后同样实现了示例 8-3 中的结果。

我们可以在格式化字符中间加入数字来规定最大的宽度。以下为几个示例及说明。

- %3d：表示输出一个宽度为 3 的有符号整数。
- %5.2f：表示输出一个宽度为 5 的浮点数，其中的小数位为 2 位，整数位为 2 位，因为小数点会占据一个宽度的位置。
- %.3f：表示要求格式化后的值的小数位为 3 位（浮点数可以不指定其总体的宽度而只指定其小数位的宽度）。
- %3.5s：表示要求格式化的字符串在 3～5 的宽度之间，第 5 个字符以后的部分将会被舍弃。
- %05s：表示要求格式化的字符串的宽度为 5，不足的位以 0 补充。

注意：如果字符串的长度或整型数位数超过说明的宽度，将按其实际长度输出；对于浮点数，若整数部分位数超过了说明的整数位宽度，将按实际整数位输出，如小数部分位数超过了说明的小数位宽度，则按说明的宽度四舍五入后输出。

【示例 8-5】以下代码演示在格式化字符中间加入数字，来限制格式化后的宽度。

```
<?php
    $num=1234.56789;          //定义一个浮点数
```

```
    $str='Hello';              //定义一个字符串
    //格式化输出
    printf('输出 3 位小数变量$num 的值：%.3f，输出字符串变量$str 的 3 个字符：
           %0.3s',$num,$str);
?>
```

代码运行结果如图 8.5 所示。

以上的内容就讲解了格式化字符的大部分知识。其他 3 个函数的格式化字符使用方法都相同，下面简要介绍一下 sprintf()函数的使用。

【示例 8-6】 以下代码演示 sprintf()函数的使用方法。

```
<?php
    $num=1234.56789;                   //定义一个浮点数
    $res=sprintf('%.2f',$num);         //使用变量接收格式化后的字符串
    echo $res;                         //输出变量值
?>
```

代码运行结果如图 8.6 所示。



图 8.5　运行结果

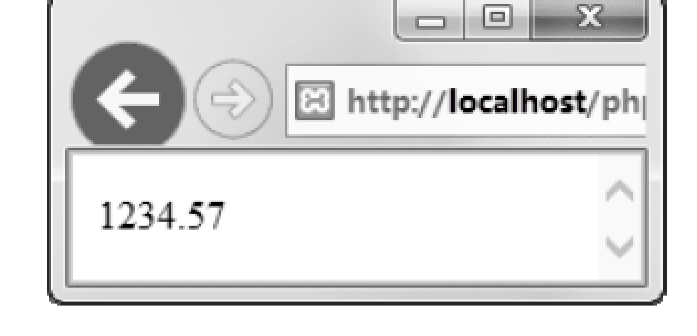


图 8.6　运行结果

其他两个函数的使用方式与 printf()函数和 sprint()函数类似，这里不再详细介绍。

8.2 去除字符

本节我们主要讲解去除字符串中的空格和 HTML 标签，这两种字符是在实际应用中常常控制的字符（串）。例如在注册或者登录系统中，我们就可以将用户输入的账号名称中的空格去掉，这样就使得用户在无意间输入了空格也可以正常登录。在留言系统中的 HTML 标签如果不被去除，则有可能被恶意用户利用 HTML 代码攻击。

8.2.1 去除空格

PHP 中提供了专门用于去除空格的 3 个函数，它们的原型如下：

```
string trim ( string $str [, string $charlist ] )
string ltrim ( string $str [, string $charlist ] )
string rtrim ( string $str [, string $charlist ] )
```

trim()函数用于删除字符串左右两侧的空字符（或者其他字符）；ltrim()函数用于删除字符串左侧的空字符（或者其他字符）；rtrim()函数用于删除字符串右侧的空字符（或者其他字符）。参数 str 即为需要处理的字符串；可选参数 charlist 用于指定想要删除的字符列表，可以使用“..”指定一个范围。几个使用指定参数范围的使用形式如下：

```
trim($str,'str')            //去除字符串$str 两侧的 str 字符串
ltrim($str,'a..z')          //去除字符串$str 左侧的小写字母
```

```
rtrim($str,'1..5')        //去除字符串$str 右侧的 1~5 之间的数字
```

在不使用可选参数的情况下，这 3 个函数会去除以下字符。

- “ ” (ASCII 32 (0x20))：普通空白符；
- “\t” (ASCII 9 (0x09))：制表符；
- “\n” (ASCII 10 (0x0A))：换行符；
- “\r” (ASCII 13 (0x0D))：回车符；
- “\0” (ASCII 0 (0x00))：NUL 空字节符；
- “\x0B” (ASCII 11 (0x0B))：垂直制表符。

【示例 8-7】以下代码演示去除空字符函数的用法。

```
<?php
    $str1='   hello';           //定义开头有 3 个空格的字符串
    $str2='hello    ';          //定义结尾有一个制表符的字符串
    $str3='
password';                      //定义开头有换行的字符串
    echo '去除空字符前：<br />';
    echo $str1.'<br />'.$str2.'<br />'.$str3.'<br />';
                                //输出去除空字符前的字符串
    echo '去除空字符后：<br />';
    //处理字符串
    $str1=ltrim($str1);
    $str2=rtrim($str2);
    $str3=ltrim($str3);
    echo $str1.'<br />'.$str2.'<br />'.$str3.'<br />';
                                //输出去除空字符后的字符串
?>
```

代码运行结果如图 8.7 所示。从运行结果看不出明显的差别，因为 HTML 会将多个连续的空格当做一个空格来显示，会将制表符省略。我们可以通过查看源代码来查看，如图 8.8 所示。



图 8.7　运行结果



图 8.8　网页源代码

从源代码第一行可以看出，两个 hello 单词一侧都有空隙，那就是空格和制表符；而 password 单词则在第 2 行代码中显示，这是换行符的作用。在经过处理后，这 3 个单词均在一行中显示而且左右均没有空隙，这就说明空字符都被去除。

我们还可以通过使用 strlen()这个函数来输出字符串的长度，以使字符串处理的结果更加明显。Strlen()函数的原型如下：

```
int strlen ( string $string )
```

参数 string 即为需要计算长度的字符串；函数的返回值即为字符串的长度。

【示例 8-8】以下代码演示使用 strlen()函数计算字符串的长度。

```
<?php
    $str1='    hello';        //定义一个开头有 4 个空格的字符串
    $str2='hello    ';        //定义一个结尾有一个制表符的字符串
    $str3='
password';                    //定义一个开头有换行的字符串
    $str4='h e ll o';         //定义一个中间有 3 个空格的字符串
    //获取字符串的长度并输出
    echo 'str1 字符串长度为: '.strlen($str1);
    echo '<br />str2 字符串长度为: '.strlen($str2);
    echo '<br />str3 字符串长度为: '.strlen($str3);
    echo '<br />str4 字符串长度为: '.strlen($str4);
?>
```

代码运行结果如图 8.9 所示。

下面我们再来使用去除空字符函数，去除指定的字符或者字符范围。

【示例 8-9】以下代码演示使用去除空字符串函数，去除指定的字符串或者字符串范围。

```
<?php
    //定义多个字符串
    $str1='whello12x';
    $str2='3462hello';
    $str3='xayABCaxy';
    $str1=trim($str1,'a..z');       //去除字符串两侧的小写字母
    $str2=ltrim($str2,'0..9');      //去除字符串左侧的数字
    $str3=trim($str3,'axy');        //去除字符串两侧的 a, x, y 字母
    echo "\$str1 处理后为: {$str1}";
    echo "<br />\$str2 处理后为: {$str2}";
    echo "<br />\$str3 处理后为: {$str3}";
?>
```

代码运行结果如图 8.10 所示。



图 8.9　运行结果



图 8.10　运行结果

从运行结果可以得知，使用去除空字符函数还可以完成一些简单的去除其他字符的功能，但是这些函数主要被用来去除字符串中的空字符。如去除其他字符，应该使用其他更合适的函数。

8.2.2　去除 HTML 和 PHP 标签

在实际的项目中，用户的输入部分要经过非常严格的判断，最基本的判断就是去除可以让用户执行恶意代码的可能。那么我们所需要做的就是限制 HTML 标签和 PHP 标签。

在 PHP 中可以使用 strip_tags()函数去除 HTML 和 PHP 标签中的内容，它的原型如下：

```
string strip_tags ( string $str [, string $allowable_tags ] )
```

参数 str 即为需要处理的字符串；参数 allowable_tags 为允许保留的字符列表。函数运行后返回处理后的字符串。

【示例 8-10】以下代码演示使用 echo()函数输出一些 HTML 代码。

```
01  <?php
02      $html=<<<HTM
03      <title>PHP 输出 HTML 代码</title>
04      <body>
05          <a href=#>转向一个其他地址的链接</a>
06      </body>
07  HTM;
08      echo $html;
09  ?>
```

注意：代码中使用的“<<<HTM…HTM”形式为字符串输出的定界符形式，包围在其中的特殊字符不需要经过转码即可正确输出，常用于大篇幅的字符串输出。定界符的名称（这里为 HTM）可以自己定义，但是结束的标记（这里为 HTM，在代码第 7 行）必须在一个新行中而且前面不可以有包括空格的任何字符。

代码运行结果如图 8.11 所示。

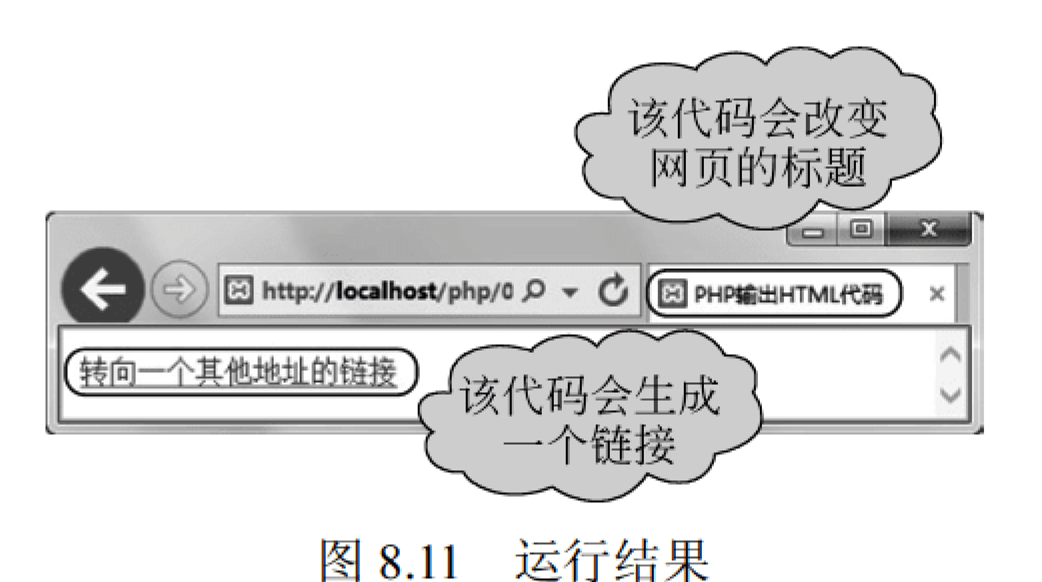


图 8.11　运行结果

从运行结果可以得知，PHP 代码中的 HTML 代码并没有被以字符串的形式输出，而是被浏览器解析后输出。那么我们就可以将标签中的内容删除，以使这些字符串不能被浏览器解析输出。

【示例 8-11】以下代码演示使用 strip_tags()函数去除字符串中的 HTML 标签内容。

```
<?php
    $html=<<<HTM
    <title>PHP 输出 HTML 代码</title>
    <body>
        <a href=#>转向一个其他地址的链接</a>
    </body>
HTM;
    $html=strip_tags($html);            //去除字符串中的 HTML 标签
    echo $html;
?>
```

代码运行结果如图 8.12 所示。



图 8.12　运行结果

从运行结果可以看到，HTML 代码没有被浏览器解析，这就是 strip_tags()函数将 HTML 标签的内容去除而保留标签外的字符串。当然，我们还可以使用 strip_tags()函数的第 2 个参数来指定不被去除的标签。

【示例 8-12】以下代码演示使用 strip_tags()函数指定不去除的 HTML 标签。

```
<?php
    $html=<<<HTM
    <title>PHP 输出 HTML 代码</title>
    <body>
        <a href=#>转向一个其他地址的链接</a>
    </body>
HTM;
    $html=strip_tags($html,'<title>');     //指定不去除<title>标签
    echo $html;
?>
```

代码运行结果如图 8.13 所示。



图 8.13　运行结果

从运行结果可以看到，该代码在运行后会改变网页的标题，这是由于我们将控制网页标题的 HTML 标签设为了例外，而其他标签仍会被删除。

在 PHP 中，有完整的开始和结束标签的 PHP 代码在 PHP 代码中不会被输出，也就是说，以下代码不会输出任何内容：

```
<?php
    echo "<?php echo 'hello!' ?>";
?>
```

这是由 PHP 解析器决定的。没有结尾标签的 PHP 代码是可以被输出的，但是这些代码可以被 PHP 正确解析执行，因此我们应该阻止这种潜在的危险。使用 strip_tag()函数可以将 PHP 代码从字符串中去除。

【示例 8-13】以下代码演示使用 strip_tags()函数去除字符串中的 PHP 代码。

```
<?php
    //定义一段包含 PHP 代码的字符串
    $php=<<<PHP
    这里是 PHP 代码的开始
    <?php
        echo "hello!";
```

```
PHP;
    $php=strip_tags($php);      //去除 PHP 代码
    echo $php;
?>
```

代码运行结果如图 8.14 所示。



图 8.14　运行结果

从运行结果可以看出，strip_tags()函数去除了 PHP 代码而保留了开头的字符。需要注意的是，php 标签“<?”之后的内容都将被去除。

8.3　字符串转换

有时候用户只是想展示一段 HTML 代码，那么我们就可以使用字符串转换函数来将一些字符转换，实现输出 HTML 代码。本节我们主要讲解一些常用的字符串转换函数。

8.3.1　大小写转换

大小写转换是比较简单和常用的字符转换函数。下面我们就来简单介绍 4 个大小写转换的函数，它们的原型如下：

```
string ucfirst ( string $str )
string ucwords ( string $str )
string strtoupper ( string $str )
string strtolower ( string $str )
```

这些函数均只接受一个必须的参数，str 参数即为需要处理的字符串。Ucfirst()函数用于将字符串的首字母转换为大写；ucwords()函数用于将字符串中每个单词的首字母转换为大写；strtoupper()函数用于将字符串转换为大写；strtolower()函数用于将字符串转换为小写。

【示例 8-14】以下代码演示使用大小写转换函数的效果。

```
<?php
    $str='PHP is a very good programming language.';   //定义一个字符串
    echo "未经处理的字符串：<br />{$str}";
    //分别使用各个函数处理字符串并输出处理后的结果
    $str=strtolower($str);
    echo "<br />使用 strtolower 函数将字符串转换为小写：<br />{$str}";
    $str=ucfirst($str);
    echo "<br />使用 ucfirst 函数将字符串首字母转换为大写：<br />{$str}";
    $str=ucwords($str);
    echo "<br />使用 ucwords 函数将字符串每个单词的首字母转换为大写：
                                                          <br /> {$str}";
    $str=strtoupper($str);
    echo "<br />使用 strtoupper 函数将字符串转换为大写：<br />{$str}";
?>
```

代码运行结果如图 8.15 所示。



图 8.15　运行结果

这些函数的理解和使用都比较简单，读者可以在实际的应用中依据情况使用这些函数，以增加一些便利。

8.3.2　换行转换

换行转换是为 HTML 制定的一个函数，在编程语言中换行通常使用“\n”来表示。PHP 是专门针对 Web 开发而设计的编程语言，但是浏览器不会将这种格式的换行解析出来。因此 PHP 就专门提供了 nl2br()函数将换行自动转换为浏览器可以解析的“
”。nl2br()函数的原型如下：

```
string nl2br ( string $string [, bool $is_xhtml = true ] )
```

参数 string 即为需要处理的字符串；可选参数 is_xhtml 用来指定是否使用 xhtml 兼容的换行符号。

【示例 8-15】以下代码演示使用 nl2br()函数自动将编程语言中的换行符，转换为浏览器可以识别的换行符。

```
01  <?php
02      $str="h
03      t
04      m
05      l";                         //定义一个多处换行的字串
06      echo "未处理前的输出形式：<br />{$str}";
07      $str=nl2br($str);           //使用 nl2br()函数处理字符串
08      echo "<br />处理后的输出形式：<br />{$str}";
09  ?>
```

以上代码运行结果如图 8.16 所示。结合代码从运行结果中就可以看出使用函数后的效果。但是在代码中我们并没有显式地见到“\n”换行符，那是因为它已经被隐式地添加到了第 3～5 行代码的开头。我们也可以在一行中显式地使用“\n”来换行。

【示例 8-16】以下代码演示在字符串中显式地使用“\n”换行符，并使用 nl2br()函数将其转换为浏览器可以解析的“
”符号。

```
<?php
    $str="h\nt\nm\nl";          //定义一个多处换行的字符串
    echo "未处理前的输出形式：<br />{$str}";
```

```
    $str=nl2br($str);        //使用 nl2br()函数处理字符串
    echo "<br />处理后的输出形式：<br />{$str}";
?>
```

代码运行结果如图 8.17 所示。



图 8.16　运行结果



图 8.17　运行结果

从运行结果可以看到，该示例中的代码实现了同示例 8-15 代码相同的功能。

8.3.3　HTML 相关转换

HTML 相关的转换通常用于将 HTML 中的指定字符转换为实体，以防止被浏览器解析，这样就可以在网站中安全地让用户输入一些带有 HTML 标签的字符。在 PHP 中常用的转换 HTML 相关转换函数有 htmlspecialchars()和 htmlentities()函数，它们的原型如下：

```
string htmlspecialchars ( string $string [, int $flags = ENT_COMPAT |
ENT_HTML401 [, string $encoding = 'UTF-8' [, bool $double_encode = true ]]] )
string htmlentities ( string $string [, int $flags = ENT_COMPAT | ENT_HTML401
[, string $encoding = 'UTF-8' [, bool $double_encode = true ]]] )
```

参数 string 为需要处理的字符串；可选参数 flags 用于规定如何处理引号；可选参数 encoding 用于规定使用的字符集。可选参数我们通常使用默认的即可。

htmlspecialchars()函数与 htmlentities()函数的区别就在于，htmlspecialchars()函数会把指定的符号转换为实体，而 htmlentities()函数会把所有可以转换的字符都转换为实体。htmlspecialchars()函数的转换规则如下。

- ❑ &（和号）：转换为&amp。
- ❑ "（双引号）：没有设置 ENT_NOQUOTES 选项时转换为&quot。
- ❑ '（单引号）：没有设置 ENT_QUOTES 选项时转换为&#039。
- ❑ <（小于）：转换为&lt。
- ❑ >（大于）：转换为&gt。

下面我们首先使用 PHP 输出一段未经处理的 HTML 代码。

【示例 8-17】以下代码演示使用 PHP 输出未经处理的 HTML 代码。

```
<?php
    //定义一个 HTML 代码字符串
    $str=<<<HTM
    <a href=#><b><i>到一个网址的链接</i></b></a>
HTM;
    echo $str;
?>
```

代码运行结果如图 8.18 所示。该段示例代码运行后输出了一个链接，而且链接的字体使用加粗和倾斜的效果，共使用了 3 种 HTML 标签。

【示例 8-18】以下代码演示使用 htmlspecialchars()函数将 HTML 代码字符串处理后并输出。

```
<?php
    //定义一个 HTML 代码字符串
    $str=<<<HTM
    <a href=#><b><i>到一个网址的链接</i></b></a>
HTM;
    $str=htmlspecialchars($str);        //将字符串中指定的字符进行转换
    echo $str;
?>
```

代码运行结果如图 8.19 所示。



图 8.18　运行结果

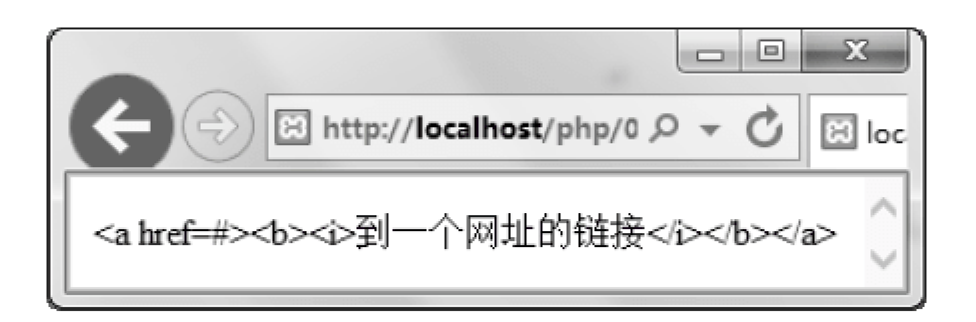


图 8.19　运行结果

从运行结果可以看到，我们将代码 str 变量中的字符完整地输出在了浏览器中。可以通过查看源代码来验证 htmlspecialchars()函数的转换效果，如图 8.20 所示。

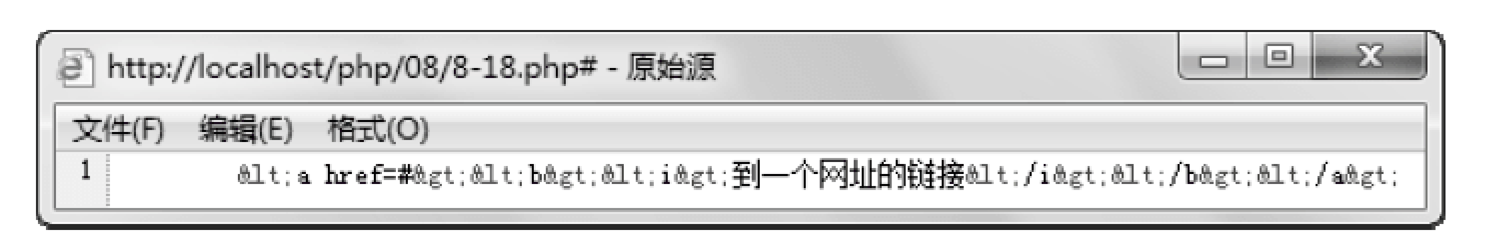


图 8.20　网页源代码

可以看到，字符串中的大于号和小于号都被进行了转换，因此浏览器就不会进行解析。

【示例 8-19】以下代码演示使用 htmlentities()函数处理示例 8-17 中的字符串。

```
<?php
    //定义一个 HTML 代码字符串
    $str=<<<HTM
    <a href=#><b><i>到一个网址的链接</i></b></a>
HTM;
    $str=htmlentities($str);        //将字符串中指定的字符进行转换
    echo $str;
?>
```

代码运行结果如图 8.21 所示。



图 8.21　运行结果

可以看到，运行的结果同示例 8-18 相同。我们同样也查看该网页的源代码，如图 8.22 所示。



图 8.22　网页源代码

从源代码中也可以看到，被转换的字符同示例 8-18 是相同的，那么这两个函数到底有什么差别呢？差别就在于 htmlspecialchars()函数只支持将我们列出的那些字符进行转换，而 htmlentities()函数支持转换的字符更多，例如希腊字母。这里就不详细介绍了，有兴趣的读者可以自己尝试一些特殊的字符来对比。

8.4　查找与替换字符串

字符串查找与替换理解起来十分容易，这也是在实际应用中常用的字符串处理类型。本节就介绍一些常用的字符串查找和替换的函数，以使读者在实际项目中根据不同的环境来应用这些函数。

8.4.1　字符串查找

字符串查找函数大致分为 3 类，分别是返回指定字符串第一次出现的位置、返回字符串最后一次出现的位置及返回字符串的子串。下面我就来分别介绍这 3 类函数。

1. strops()和 stripos()函数

strops()函数用于返回指定字符串首次出现的位置，stripos()函数是 strpos()函数不区分大小写的版本，这两个函数的原型如下：

```
int strpos ( string $haystack , mixed $needle [, int $offset = 0 ] )
int stripos ( string $haystack , string $needle [, int $offset = 0 ] )
```

参数 haystack 即为被查找的字符串；参数 needle 即为需要查找的字符串；可选参数 offset 用来设置偏移量，该偏移量是相对于 haystack 字符串的起始位置而言的。

【示例 8-20】以下代码演示在字符串中查找指定字符第一次出现的位置。

```
<?php
    $str='password';                  //定义一个字符串
    $position=strpos($str,'w');       //查找字母 s 第一次出现的位置
    echo '字母 w 的位置是'.$position;
?>
```

注意：字符串的位置是从 0 开始计算的，即第一个字符串位置为 0，依次类推。

代码运行结果如图 8.23 所示。下面我们为 strpos()函数的可选参数 offset 传入值，以控制搜索开始的位置。

【示例 8-21】以下代码演示控制 strpos()函数开始查找的位置。

```
<?php
    $str='password';                      //定义一个字符串
    $position=strpos($str,'s');           //查找字母 s 第一次出现的位置
    echo '字母 s 的位置是'.$position;
```

```
    $position=strpos($str,'s',3);        //控制查找开始的位置
    echo '<br />控制查找位置后字母 s 的位置是'.$position;
?>
```

代码运行结果如图 8.24 所示。从运行结果中可以看到，两次查找相同的字母但是位置是不同的，这就是控制偏移量的结果。stripos()函数的使用方法同 strpos()函数类似，我们以一个简单的示例来演示。

【示例 8-22】以下代码演示使用 stripos()函数进行不区分大小写地查找字符。

```
<?php
    $str='password';                     //定义一个字符串
    $position=strpos($str,'S');          //查找字母 s 第一次出现的位置
    echo '字母 S 的位置是'.$position;
    $position=stripos($str,'S');         //控制查找开始的位置
    echo '<br />控制查找位置后字母 S 的位置是'.$position;
?>
```

代码运行结果如图 8.25 所示。



图 8.23　运行结果



图 8.24　运行结果

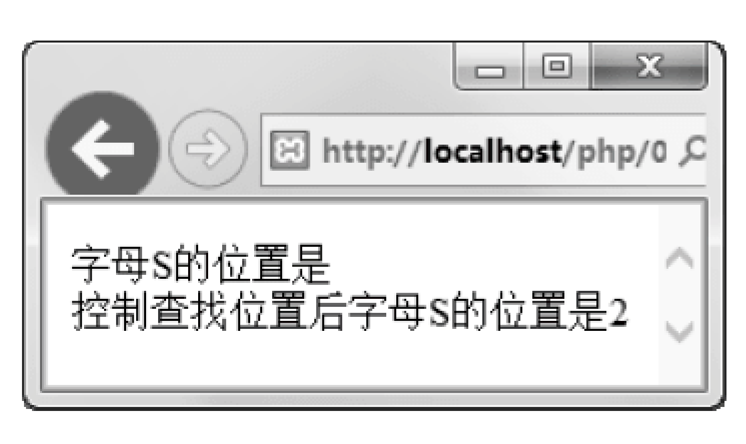


图 8.25　运行结果

从运行结果的第一行可以看出，strpos()函数并没有找到指定的字符，因此不会返回字符的位置。而 stripos()函数由于不区分查找字符的大小写，因此返回了字符的位置。

2. strrpos()和 strripos()函数

strrpos()和 strripos()函数是与 strpos()和 stripos()对应的函数。strrpos()函数用于返回指定字符最后出现在字符串中的位置；strripos()函数则是 strrpos()函数的不区分大小写的版本，这两个函数的原型如下：

```
int strrpos ( string $haystack , string $needle [, int $offset = 0 ] )
int strripos ( string $haystack , string $needle [, int $offset = 0 ] )
```

参数 haystack 为被查找的字符串，参数 needle 为需要查找的字符，如果为字符串则只取第一个字符；可选参数 offset 用来设置开始查找的位置。

【示例 8-23】以下代码演示使用 strrpos()函数和 strripos()函数，查找字符在指定字符串中最后出现的位置。

```
<?php
    //定义两个字符串
    $str='password';
    $str1='Hello PHP!';
    $position=strrpos($str,'s');         //不区分大小写判断
    echo "字母 s 在{$str}中最后出现的位置是{$position}";
    $position=strripos($str1,'h');       //区分大小写判断
    echo "<br />字母 h 在{$str1}中最后出现的位置是{$position}";
?>
```

代码运行结果如图 8.26 所示。需要注意，空格是字符串，因此会被计算为一个位置。这两个函数的使用和理解都非常简单，这里不再详细介绍。

3. strrchr()、strstr()和 stristr()函数

除了以上一些返回字符所在位置的函数之外，还有一些类似的函数，这些函数会查找到指定的字符并返回相对这个字符串位置的一些字符串。我们首先来看 strrchr()函数，它的原型如下：

```
string strrchr ( string $haystack , mixed $needle )
```

该函数会从参数 haystack 中找到 needle 参数，并返回从此参数到 haystack 结尾的所有字符串。

【示例 8-24】以下代码演示 strrchr()函数的使用方法及效果。

```
<?php
    $str='PHP is a very good programming language!';    //定义一个字符串
    echo strrchr($str,'p');     //输出最后一次出现字母 p 的位置到字符串结尾的所有字符串
?>
```

代码运行结果如图 8.27 所示。



图 8.26　运行结果

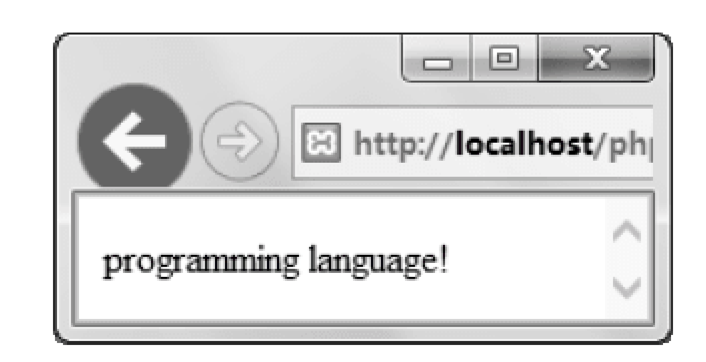


图 8.27　运行结果

与 strrchr()函数对应的函数是 strstr()函数和 stristr()函数，这两个函数会查找到指定字符串首次出现的位置，并返回从该位置到结尾的字符串，它们的原型如下：

```
string strstr (string $haystack, mixed $needle [, bool $before_needle = false ])
string stristr (string $haystack, mixed $needle [, bool $before_needle = false ])
```

参数 haystack 即为被查找的字符串；参数 needle 为需要查找的字符；如果可选的参数 needle 被设置为 TRUE，则返回 needle 位置之前的字符串。stristr()函数是 strstr()函数不区分大小写的版本。

【示例 8-25】以下代码演示 strstr()函数的使用方法及效果。

```
<?php
    $str='PHP is a very good programming language!';    //定义一个字符串
    echo '第一次出现字母 l 的位置到字符串结尾的所有字符串'.strstr($str,'l');
    echo '<br />第一次出现字符串的位置到字符串结尾的所有字符串
                                                  '.strstr($str,'p');
?>
```

代码运行结果如图 8.28 所示。

strstr()和 stristr()函数不仅可以匹配单个字符，也可以匹配一个字符串，这样可以使得匹配更加精确。



图 8.28　运行结果

【示例 8-26】以下代码演示使用 stristr()函数不区分大小写地匹配字符，并返回指定字符串到结尾的字符串。

```
01  <?php
02      $str='PHP is a very good programming language!';   //定义一个字符串
03      echo '不区分大小写地输出第一次出现字母 p 到结尾的字符串
                                        '.stristr($str,'p');
04      echo '<br />不区分大小写地输出第一次出现 pro 字符串到结尾的字符串
                                        '.stristr($str,'pro');
05  ?>
```

代码运行结果如图 8.29 所示。

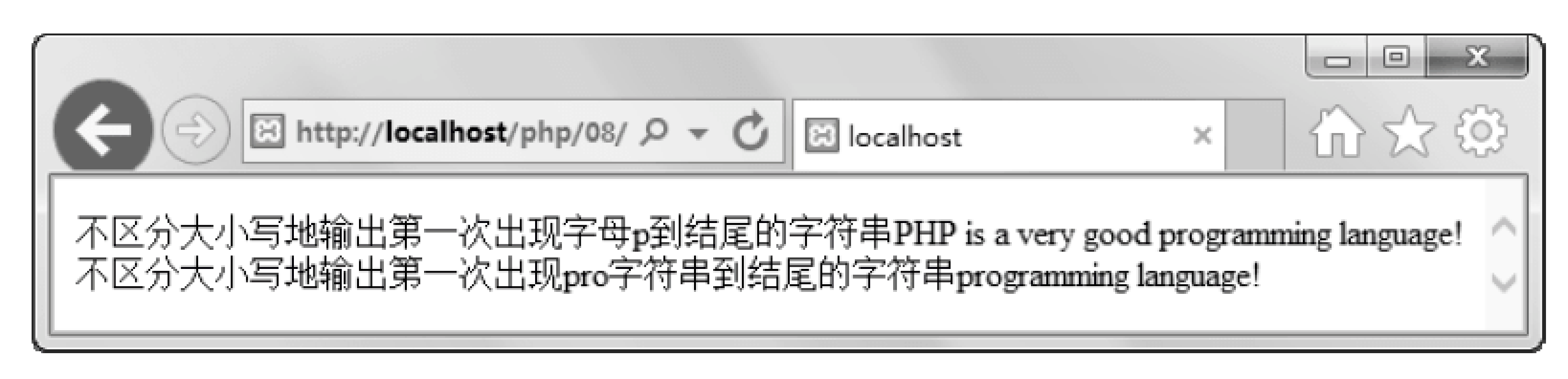


图 8.29　运行结果

在本示例中需要注意的是，第 4 行代码中使用 stristr()函数匹配的是 pro 字符串，因此虽然字符串的开头字母即为 P 但是该函数会完全匹配字符串。

4. substr()函数

substr()函数用来从一个字符串指定的位置开始，返回指定长度的一段字符串。它的函数原型如下：

```
string substr ( string $string , int $start [, int $length ] )
```

参数 string 即为源字符串；参数 start 即为开始的位置；可选的参数用来规定返回的字符串的长度，默认为到字符串的结尾。

【示例 8-27】以下代码演示使用 substr()函数返回指定的字符串。

```
<?php
    $str='programming';         //定义一个字符串
    $res=substr($str,0,4);      //处理字符串
    echo '返回从 0 开始 4 个宽度的字符串'.$res;
    $res=substr($str,7);        //处理字符串
    echo '<br />返回从 8 开始到结尾的字符串'.$res;
?>
```

代码运行结果如图 8.30 所示。需要注意的是，substr()函数的 start 和 length 参数都可以为负数。如果 start 参数为负数，则从字符串的结尾往前数；如果 length 参数为负数，则将从 start 指定的位置到结尾的字符串中减去正数 length 个字符。

【示例 8-28】以下代码演示 substr()函数的参数为负数时的执行效果。

```
<?php
    $str='programming';          //定义一个字符串
    $res=substr($str,-4);        //处理字符串
    echo "返回{$str}字符串的最后 4 个字符{$res}";
    $res=substr($str,0,-4);      //处理字符串
    echo "<br />返回{$str}字符串除去后 4 个字符{$res}";
?>
```

代码运行结果如图 8.31 所示。



图 8.30 运行结果



图 8.31 运行结果

读者应该熟练掌握这个函数的用法，以在实际项目中灵活运用。

8.4.2 字符串替换

PHP 提供了非常灵活的字符串替换函数，本节中我们将主要介绍 str_replace()和 substr_replace()函数。

1. str_replace()函数

str_replace()函数用于替换字符串中指定的字符串，它的原型如下：

```
mixed str_replace(mixed $search, mixed $replace, mixed $subject [,int &$count])
```

参数 search 为目标的字符串；参数 replace 为要替换为的字符串；参数 subject 为源字符串；可选参数 count 接受一个变量，用于统计替换发生的次数。我们可以看到，str_replace()函数的返回值和函数的必须参数的类型都为混合类型，这就使得该函数的使用可以非常灵活、多变。首先来看一个该函数最简单的使用示例。

【示例 8-29】以下代码演示 str_replace()函数最简单的使用形式。

```
<?php
    $str='Hello world!';                        //定义源字符串
    $search='o';                                //定义将被替换的字符
    $replace='O';                               //定义替换的字符串
    $res=str_replace($search,$replace,$str);    //使用函数处理字符串
    echo "{$str}替换后的效果为：<br />{$res}";
?>
```

代码运行结果如图 8.32 所示。这段示例代码很好理解，就是将字符串中的小写字母 o 替换为大写字母 O。我们可以使用可选参数来获取替换发生的次数。

【示例 8-30】以下代码演示使用 str_replace()函数的可选参数，获取替换发生的次数。

```
<?php
    $str='Hello world!';                    //定义源字符串
```

```
    $search='o';                                    //定义将被替换的字符
    $replace='O';                                   //定义替换的字符串
    $res=str replace($search,$replace,$str,$count);
                                                    //使用可选参数记录替换次数
    echo "{$str}替换后的效果为: <br />{$res}";
    echo "<br />替换发生的次数为{$count}";
?>
```

代码运行结果如图 8.33 所示。从运行结果可以看到，在代码中通过为可选参数传递一个变量，从而统计了替换发生的次数。接下来演示 search 参数为数组时的运行效果。

【示例 8-31】以下代码演示 str_replace()函数的搜索参数为数组时的运行效果。

```
<?php
    $str='Hello world!';                            //定义源字符串
    $search=array('o','l','w');                     //定义将被替换的字符数组
    $replace='O';                                   //定义替换的字符串
    $res=str replace($search,$replace,$str);        //使用函数处理字符串
    echo "{$str}替换后的效果为: <br />{$res}";
?>
```

代码运行结果如图 8.34 所示。



图 8.32　运行结果



图 8.33　运行结果



图 8.34　运行结果

这个示例也是比较好理解的，就是将所有搜索字符串数组中的字符，都替换为大写字母 O。下面接着来看 replace 参数也为数组时的运行效果。

【示例 8-32】以下代码演示 str_replace()函数的 replace 参数也为数组时的运行效果。

```
<?php
    $str='Hello world!';                            //定义源字符串
    $search=array('o','l','w');                     //定义将被替换的字符数组
    $replace=array('1','2','3');                    //定义替换的字符串数组
    $res=str replace($search,$replace,$str);        //使用函数处理字符串
    echo "{$str}替换后的效果为: <br />{$res}";
?>
```

代码运行结果如图 8.35 所示。当搜索字符串和替换字符串都为数组时，str_replace()函数会把搜索数组中的对应项替换为替换数组中的对应项。示例 8-32 中的搜索项和替换项为长度相等的数组，下面我们来看搜索数组的元素少于替换数组元素的情况。

【示例 8-33】以下代码演示 str_replace()函数的 search 数组项少于 replace 数组项的情况。

```
<?php
    $str='Hello world!';                            //定义源字符串
    $search=array('o','l');                         //定义将被替换的字符数组
    $replace=array('1','2','3');                    //定义替换的字符串数组
    $res=str replace($search,$replace,$str);        //使用函数处理字符串
    echo "{$str}替换后的效果为: <br />{$res}";
?>
```

代码运行结果如图 8.36 所示。这种形式也容易理解，因为搜索数组中只有两个元素，

因此只会对应替换数组中的前两个元素，因此即使替换数组有再多的元素，也是没有任何作用的，因为需要替换的就只有两个元素。下面再来看替换数组元素比搜索数组元素少的情况。

【示例 8-34】以下代码演示 str_replace()函数的 replace 参数数组元素，比 search 参数数组元素少时的运行效果。

```
<?php
    $str='Hello world!';                        //定义源字符串
    $search=array('o','l','w');                 //定义将被替换的字符数组
    $replace=array('1','2');                    //定义替换的字符串数组
    $res=str_replace($search,$replace,$str);    //使用函数处理字符串
    echo "{$str}替换后的效果为：<br />{$res}";
?>
```

代码运行结果如图 8.37 所示。




图 8.35　运行结果



图 8.36　运行结果



图 8.37　运行结果

这种情况下，搜索参数数组比替换参数数组多出来的元素会被替换为空（NULL）。也就说，该示例中，原字符串中的字母 w 会替换为空。当然，str_replace()函数不只是可以用来替换字符，这里是为了让读者更容易看清变化而采用替换字符的形式。下面我们就演示一个简单替换字符串的形式。

【示例 8-35】以下代码演示使用 str_replace()函数替换字符串。

```
<?php
    $str='Hello world!';                        //定义源字符串
    $search=array('Hello','world','!');         //定义将被替换的字符数组
    $replace=array('Hi','PHP','!!!');           //定义替换的字符串数组
    $res=str_replace($search,$replace,$str);    //使用函数处理字符串
    echo "{$str}替换后的效果为：<br />{$res}";
?>
```

代码运行结果如图 8.38 所示。到这里为止 str_replace()函数的绝大部分使用形式我们都做了介绍，虽然每个示例都比较简单，但是如果将这么多的知识点用一个大的示例讲解，相信有很多人都不容易理解。PHP 中还有一个与 str_replace()函数对应的函数 str_ireplace()，该函数是 str_replace()函数不区分大小写的版本，它的原型如下：

```
mixed str_ireplace ( mixed $search , mixed $replace , mixed $subject [, int
&$count ] )
```

该函数的使用方法与 str_replace()函数的使用方法极为类似，这里不再详细介绍。

2. substr_replace()函数

substr_replace()函数用来替换字符串中的子串，它的原型如下：

```
mixed substr_replace ( mixed $string , string $replacement , int $start [,
```

```
int $length])
```

参数 string 为源字符串；参数 replacement 为替换的字符串；参数 start 为替换开始的位置，如果为负数则从字符串末尾向前数；可选参数 length 为正数，则表示要替换源字符串中字符的个数，为负数则表示替换 start 位置开始到距离结尾 length 长度范围内的字符；为 0 则不替换任何字符，而是将 replacement 字符插入到 start 指定的位置，默认情况下为替换所有的字符。我们首先来演示该函数最简单的使用形式。

【示例 8-36】以下代码演示 substr_replace()函数最简单的使用形式。

```
<?php
    $str='programming';                              //定义一个字符串
    $replacement='er';                               //定义替换的字符串
    $res=substr_replace($str,$replacement,8);        //处理字符串
    echo "{$str}执行替换后的效果：<br />{$res}";
?>
```

代码运行结果如图 8.39 所示。结合运行结果和代码我们很容易就可以理解，substr_replace()函数在本示例中所执行的过程为，将$str 变量字符串的第 8 个字符开始到字符串结尾的字符替换为 er。接下来我们再来演示 start 参数为负数的情况下完成示例 8-36 的效果。

【示例 8-37】以下代码演示为 substr_replace()函数的 start 参数传入负值，来实现示例 8-36 中的运行效果。

```
<?php
    $str='programming';                              //定义一个字符串
    $replacement='er';                               //定义替换的字符串
    $res=substr_replace($str,$replacement,-4);       //处理字符串
    echo "{$str}执行替换后的效果：<br />{$res}";
?>
```

代码运行结果如图 8.40 所示。




图 8.38　运行结果



图 8.39　运行结果



图 8.40　运行结果

从运行结果可以看出，该示例的运行效果与示例 8-36 的运行效果是一致的。下面我们来看当可选参数 length 被设置为正数的情况。

【示例 8-38】以下代码演示 substr_replace 含的可选参数 length 参数，被设置为正数时执行的效果。

```
<?php
    $str='programming';                              //定义一个字符串
    $replacement='er';                               //定义替换的字符串
    $res=substr_replace($str,$replacement,-4,1);     //处理字符串
    echo "{$str}执行替换后的效果：<br />{$res}";
?>
```

代码运行结果如图 8.41 所示。在代码中参数 length 被指定为 1，则函数在执行时只替换$str 字符串中的一个字符，也就是 programming 字符串中的第 8 个字符 m 被替换为字符串 er。下面来看 length 参数被设置为负数的情况。

【示例 8-39】以下代码演示将 subs_replace()函数的可选参数 length，设置为负数时的运行效果。

```
<?php
    $str='programming';                              //定义一个字符串
    $replacement='er';                               //定义替换的字符串
    $res=substr_replace($str,$replacement,-4,-1);    //处理字符串
    echo "{$str}执行替换后的效果：<br />{$res}";
?>
```

代码运行结果如图 8.42 所示。该函数在本示例中的执行过程为，将字符串 programming 从倒数第 4 个字符开始替换到倒数第一个字符。也就是说将字符 min 替换为了字符 er。接着我们来看 length 参数被设置为 0 时的情况。

【示例 8-40】以下代码演示将 substr_replace()函数的可选参数 length，设置为 0 时的运行效果。

```
<?php
    $str='programming';                              //定义一个字符串
    $replacement='er';                               //定义替换的字符串
    $res=substr_replace($str,$replacement,-4,0);     //处理字符串
    echo "{$str}执行替换后的效果：<br />{$res}";
?>
```

代码运行结果如图 8.43 所示。



图 8.41　运行结果



图 8.42　运行结果



图 8.43　运行结果

这个运行结果也是比较容易理解的，当参数 length 被设置为 0 时也就表示不替换任何字符，那么就将 replacement 参数插入到 start 参数指定的位置。也就是说将字符 er 插入到 programming 字符串的倒数第 4 个位置。

8.5　合并与拆分字符串

合并字符串用于将多个字符或者字符串合并为一个字符串；拆分字符串则是用来将一个字符串拆分为多个字符串或字符。本节就来介绍一些有关合并与拆分字符串的函数。

8.5.1　将数组和字符串之间转换

有些情况下，需要将一个字符串分割为多个子串作为数组的元素来处理。PHP 提供了简单的 str_split()函数和有更强功能的 explode()函数来完成简单的字符串到数组的转换。

1. str_split()函数

str_split()函数的原型如下：

```
array str_split ( string $string [, int $split_length = 1 ] )
```

参数 string 为需要处理的字符串；可选参数 length 用来规定每个子串的宽度，默认为一个字符。该函数的使用方法比较简单，我们通过一个简单的示例来讲解。

【示例 8-41】以下代码演示使用 str_split()函数将一个字符串分割为单个字符并作为数组的元素。

```
<?php
    $str='programming';          //定义一个字符串
    $arr=str_split($str);        //将字符串分割并传入数组
    print_r($arr);               //输出数组详细信息
?>
```

代码运行结果如图 8.44 所示。

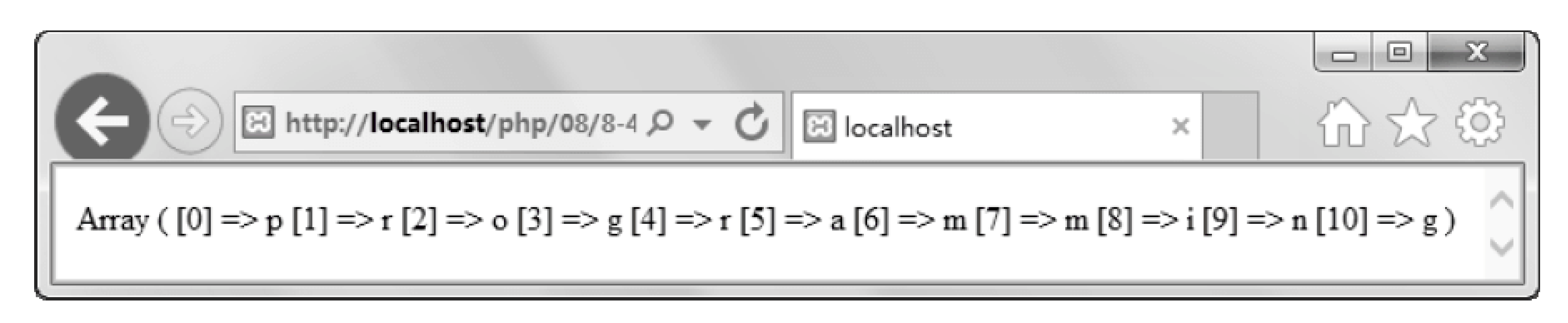


图 8.44　运行结果

从运行结果可以得知，programming 字符串被以单个字符串宽度分割到了数组中。我们可以通过设置可选的参数 length，来更改每个子串的宽度。

【示例 8-42】以下代码演示使用 str_split()函数的可选参数设置每个子串的宽度。

```
<?php
    $str='programming';          //定义一个字符串
    $arr=str_split($str,3);      //指定分割字符串的宽度
    print_r($arr);               //输出数组详细信息
?>
```

代码运行结果如图 8.45 所示。



图 8.45　运行结果

这里需要注意的是，生成字符串的最后一个元素中的字符串，可能不够 str_split()函数中指定的宽度。

2. explode()和 implode()函数

explode()函数用于使用一个字符串来分割另一个字符串，它的原型如下：

```
array explode ( string $separator , string $string [, int $limit ] )
```

参数 separator 作为分割标识的字符；参数 string 为需要被分割的字符；可选参数用于规定返回数组元素的个数，如果为正数则最后一个数组会包含所有剩余的字符；如果为负数则返回除了后面 limit 个元素之外的数组。首先来演示该函数最简单的使用形式。

【示例 8-43】以下代码演示使用 explode()函数将一个字符串分割为多个单词并作为数组的元素。

```
<?php
    $str='PHP is a very good programming language';    //定义一个字符串
    $arr=explode(' ',$str);     //使用空格分割字符串
    print_r($arr);              //输出生成的数组
?>
```

代码运行结果如图 8.46 所示。

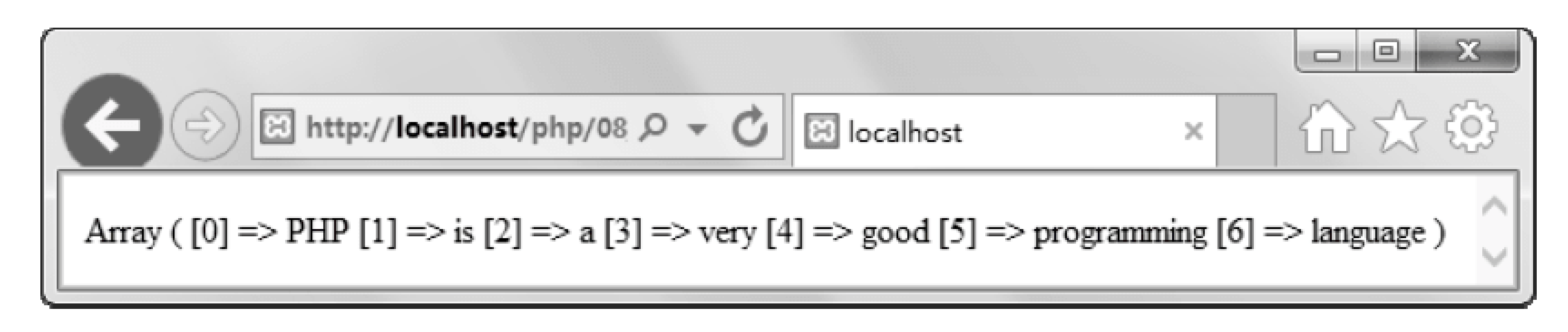


图 8.46　运行结果

从运行结果得知，每个单词都被作为了数组的元素。接下来我们来使用 limit 参数来限制生成数组元素的个数。

【示例 8-44】以下代码演示通过设置 explode()函数的 limit 参数，来限制生成的数组元素个数。

```
<?php
    $str='PHP is a very good programming language';    //定义一个字符串
    $arr=explode(' ',$str,3);   //使用空格分割字符串并限制生成数组的元素个数
    print_r($arr);              //输出生成的数组
?>
```

代码运行结果如图 8.47 所示。这里读者一定要注意的是，最后一个数组将接收全部剩余没有被分割的字符。我们接着来看将 limit 设置为负值时的执行效果。

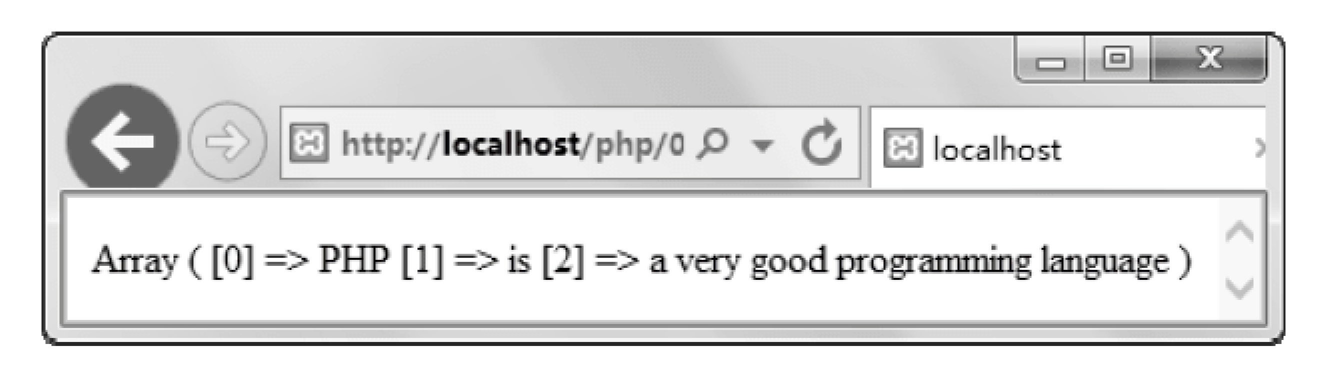


图 8.47　运行结果

【示例 8-45】以下代码演示将 explode()函数的 limit 参数设置为负数时的运行效果。

```
<?php
    $str='PHP is a very good programming language';    //定义一个字符串
    $arr=explode(' ',$str,-3);  //使用空格分割字符串并限制生成数组的元素个数
    print_r($arr);              //输出生成的数组
?>
```

代码运行结果如图 8.48 所示。

这个执行效果也是比较容易理解的。我们知道，如果不限制数组输出元素个数的情况

下，数组为一个含有 7 个元素的数组，而我们限制了 limit 参数为“-3”，则该函数会将数组的后 3 个元素省略后返回，因此就会输出一个拥有 4 个元素的数组。

我们知道，如果限制 explode()函数的 limit 为“1”的时候，函数会将整个字符串作为一个数组元素返回，但是需要注意的是，当 limit 参数被设置为 0 时，函数也会将整个字符串作为一个数组元素返回。当然我们通常不会将它设置为 0。

当我们设置的分割字符在字符串中不存在时，函数会将整个字符串作为一个数组元素返回。

【示例 8-46】以下代码演示将 explode()函数的分割数组设置为源数组中不存在的字符串时的运行结果。

```
<?php
    $str='PHP is a very good programming language';     //定义一个字符串
    $arr=explode('1',$str,0);   //使用空格分割字符串并限制生成数组的元素个数
    print_r($arr);              //输出生成的数组
?>
```

代码运行结果如图 8.49 所示。



图 8.48 运行结果

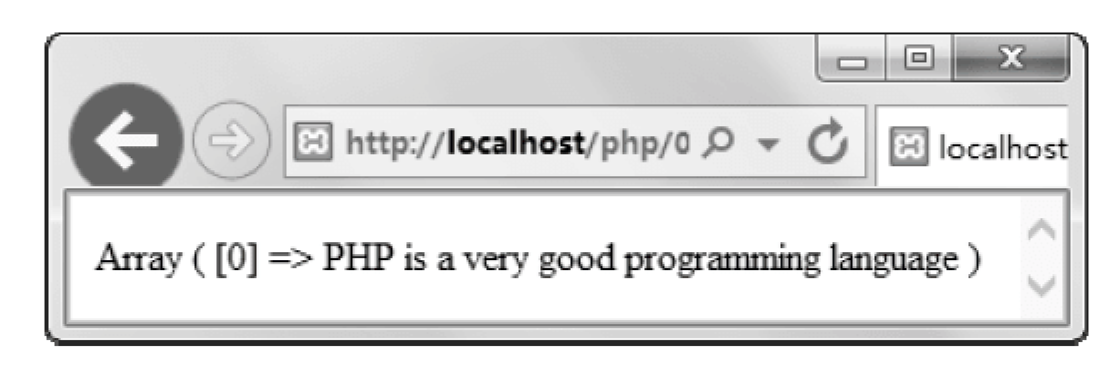


图 8.49 运行结果

与 explode()函数对应的是 implode()函数，它会将一个数组中的元素合并为一个字符串，它有两种使用形式，如下所示。

```
string implode ( string $glue , array $pieces )
string implode ( array $pieces )
```

其中的参数 pieces 即为需要被合并为字符串的数组；第一种形式中的参数 glue 为合并字符串时的连接符号。

【示例 8-47】以下代码演示使用 implode()函数将一个数组元素合并为一个字符串。

```
<?php
    $arr=array('PHP','is','a','good','programming','language');
                                //定义一个数组
    echo '该数组的元素如下：<br />';
    print_r($arr);              //首先输出数组的详细元素
    $str=implode($arr);         //使用第二种形式处理数组
    echo "<br />使用第二种形式处理后的效果：<br />{$str}";
    $str=implode(' ',$arr);    //使用第一种形式处理数组并使用空格作为连接符号
    echo "<br />使用空格作为连接符：<br />{$str}";
    $str=implode('#',$arr);    //使用第二种形式处理数组并使用#号作为连接符号
    echo "<br />使用#号作为连接符：<br />{$str}";
?>
```

代码运行结果如图 8.50 所示。

implode()函数的使用和理解都比较简单，这里就不再详细介绍。

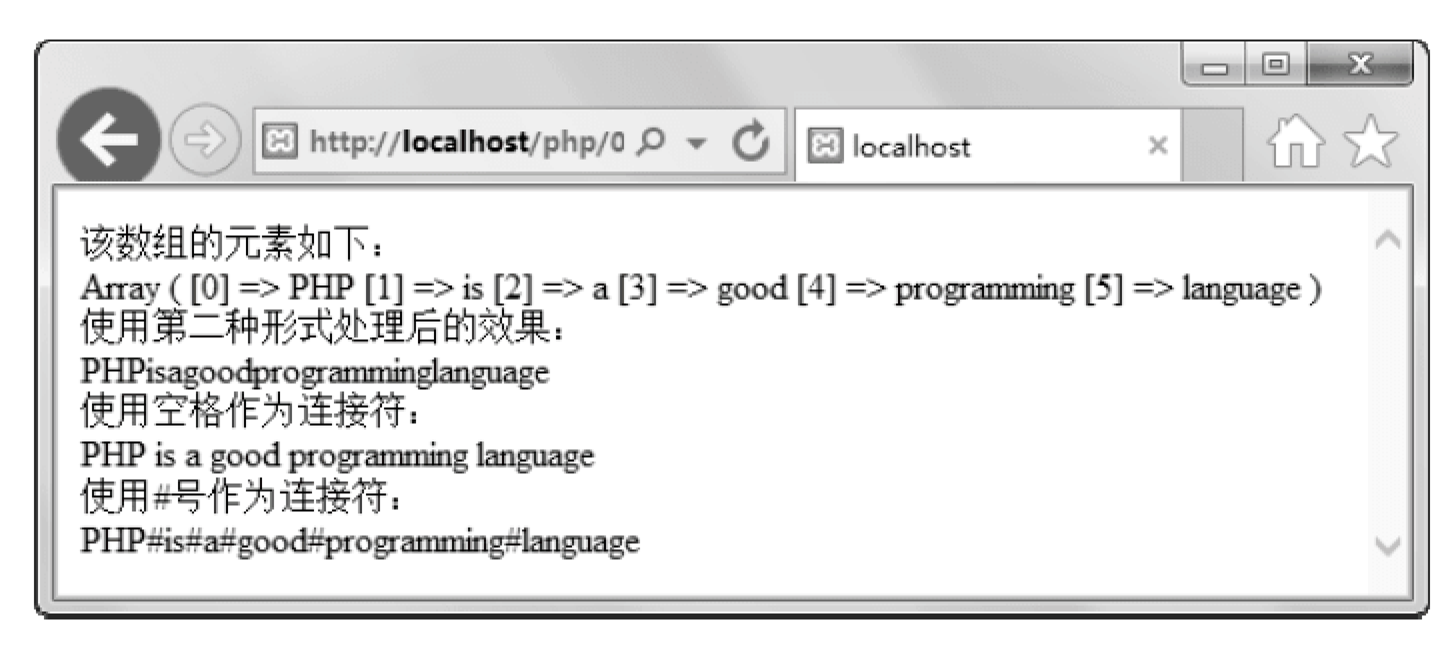


图 8.50　运行结果

8.5.2　strtok()函数

strtok()函数用于分割字符串，它有两种使用形式：

```
string strtok ( string $str , string $token )
string strtok ( string $token )
```

参数 str 为需要被分割的字符串；参数 token 为分割的标记。该函数的第一种形式只在初始化的时候使用一次，用来指定使用分割的标记。如果需要指定新的规则，则只需再次执行这种形式即可。第二种形式用来在使用第一种形式初始化分割标记后使用，每次执行均会返回新分割出的字符串。

【示例 8-48】以下代码演示使用 strtok()函数分割一个字符串。

```
01  <?php
02      $str='PHP is a good programming language';      //定义一个字符串
03      echo "字符串{$str}<hr />";        //输出字符串并在另一行输出一条横线
04      $res=strtok($str,' ');            //初始化规则并返回分割出的第一个字符串
05      echo "第一个被分割出来的字符串为：{$res}";
06      //使用第二种形式获取分割出的字符串
07      echo '<br />第二个被分割出来的字符串为：'.strtok(' ');
08      echo '<br />第三个被分割出来的字符串为：'.strtok(' ');
09      echo '<br />第四个被分割出来的字符串为：'.strtok(' ');
10  ?>
```

代码运行结果如图 8.51 所示。从代码中可以看到，第 7～9 行代码大部分相同，因此我们可以使用循环来输出剩余的字符串。

【示例 8-49】以下代码演示使用循环输出 strtok()函数分割的字符串。

```
<?php
    $str='PHP is a good programming language';      //定义一个字符串
    echo "字符串{$str}<hr />";        //输出字符串并在另一行输出一条横线
    $num=0;
    $res=strtok($str,' ');            //初始化规则并返回分割出的第一个字符串
    while($res!==FALSE){
        $num++;
        echo "第{$num}个被分割出来的字符串为：{$res}<br />";
        $res=strtok(' ');            //循环输出剩余字符串
    }
?>
```

代码运行结果如图 8.52 所示。

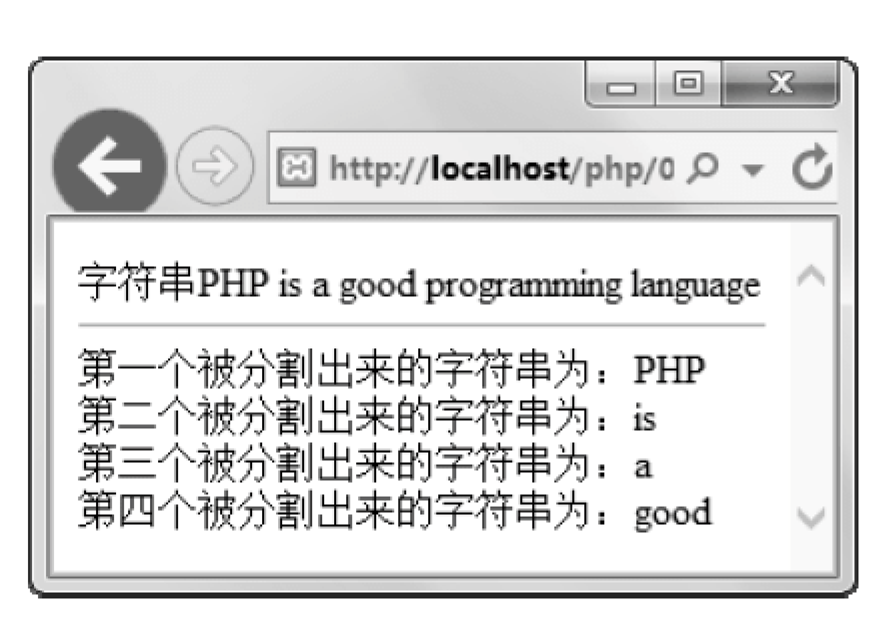


图 8.51　运行结果



图 8.52　运行结果

从运行结果可以看到，使用循环可以很便利地输出 strtok()函数分割出的字符串。但需要注意的是，将字符串处理函数的返回值作为判断条件时通常需要使用全等运算符，因为字符串处理函数可能返回 0 和空字符，而它们是不严格相等于 FALSE 的，因此有时就会导致循环被意外终止。

8.5.3　wordwrap()函数

wordwrap()函数用于在指定长度的字符串后加入换行，它的原型如下：

```
string wordwrap ( string $str [, int $width = 75 [, string $break = "\n"
[, bool $cut = false ]]] )
```

参数 str 为需要处理的字符串；可选参数 width 用来指定单行的最大宽度；可选参数 break 为达到 width 指定的宽度时插入的换行符；参数 cut 用来控制达到 width 指定宽度时是否截断单词。我们仍然沿用前面的策略，首先从最简单的形式来讲解。

【示例 8-50】以下代码演示使用 wrodwrap()函数处理字符串。

```
<?php
    $str='PHP is a widely-used general-purpose scripting language that
    is especially suited for Web development and can be embedded into
    HTML.';                                         //定义一个长的字符串
    echo "未经处理的输出：<br />{$str}";              //输出未经处理的原字符串
    $str=wordwrap($str);                            //处理字符串
    echo "<br />默认处理后的输出：<br />{$str}";      //输出处理后的字符串
?>
```

代码运行结果如图 8.53 所示。

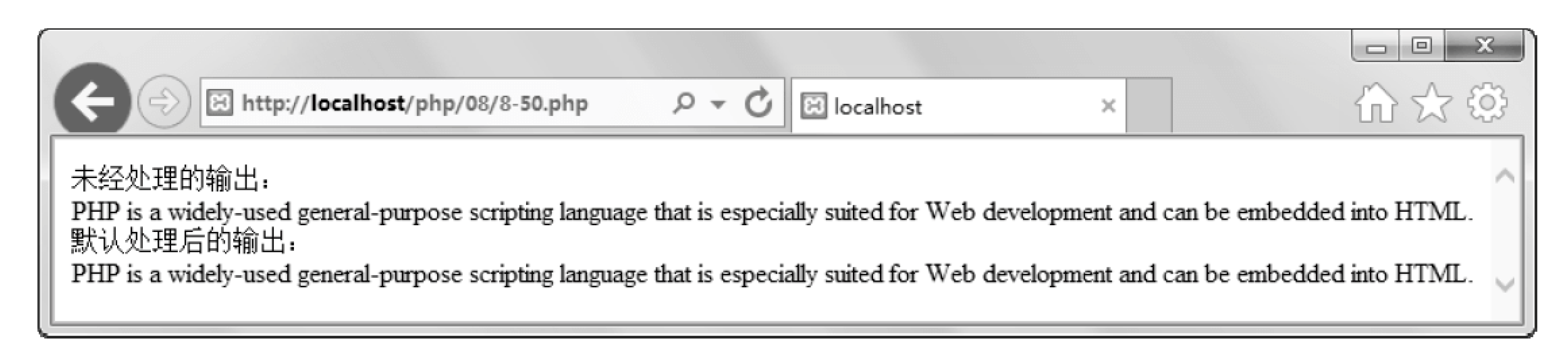


图 8.53　运行结果

从运行结果我们看不出字符处理前后差别，因为 wordwrap()函数默认加入的换行符为“\n”，我们可以从网页源代码中来看效果，如图 8.54 所示。

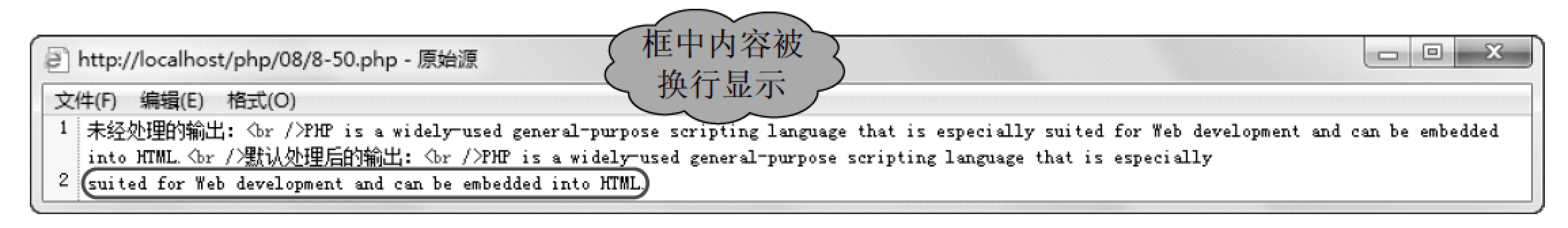


图 8.54　网页源代码

wordwrap()函数默认的截断宽度为 75，我们可以通过设置第二个可选参数来自定义宽度。

【示例 8-51】以下代码演示为 wordwrap()函数设置自定义宽度。

```
<?php
    $str='PHP is a widely-used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML.';                                //定义一个长的字符串
    echo "未经处理的输出: <br />{$str}";      //输出未经处理的原字符串
    $str=wordwrap($str,10);                   //设置最大的截断宽度为 10
    echo "<br />处理后的输出: <br />{$str}";  //输出处理后的字符串
?>
```

代码运行结果如图 8.55 所示。

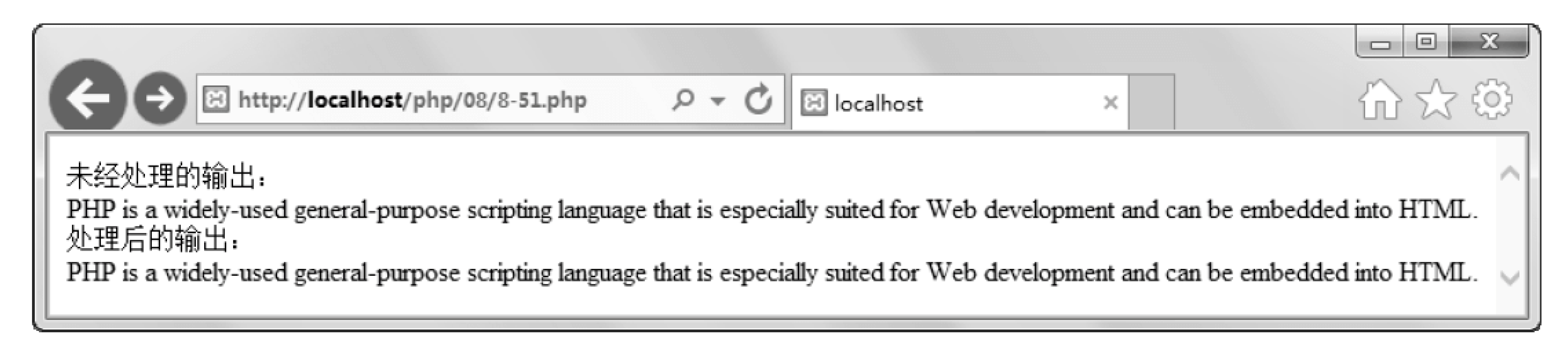


图 8.55　运行结果

我们同样需要从源代码来看该函数执行的效果，如图 8.56 所示。

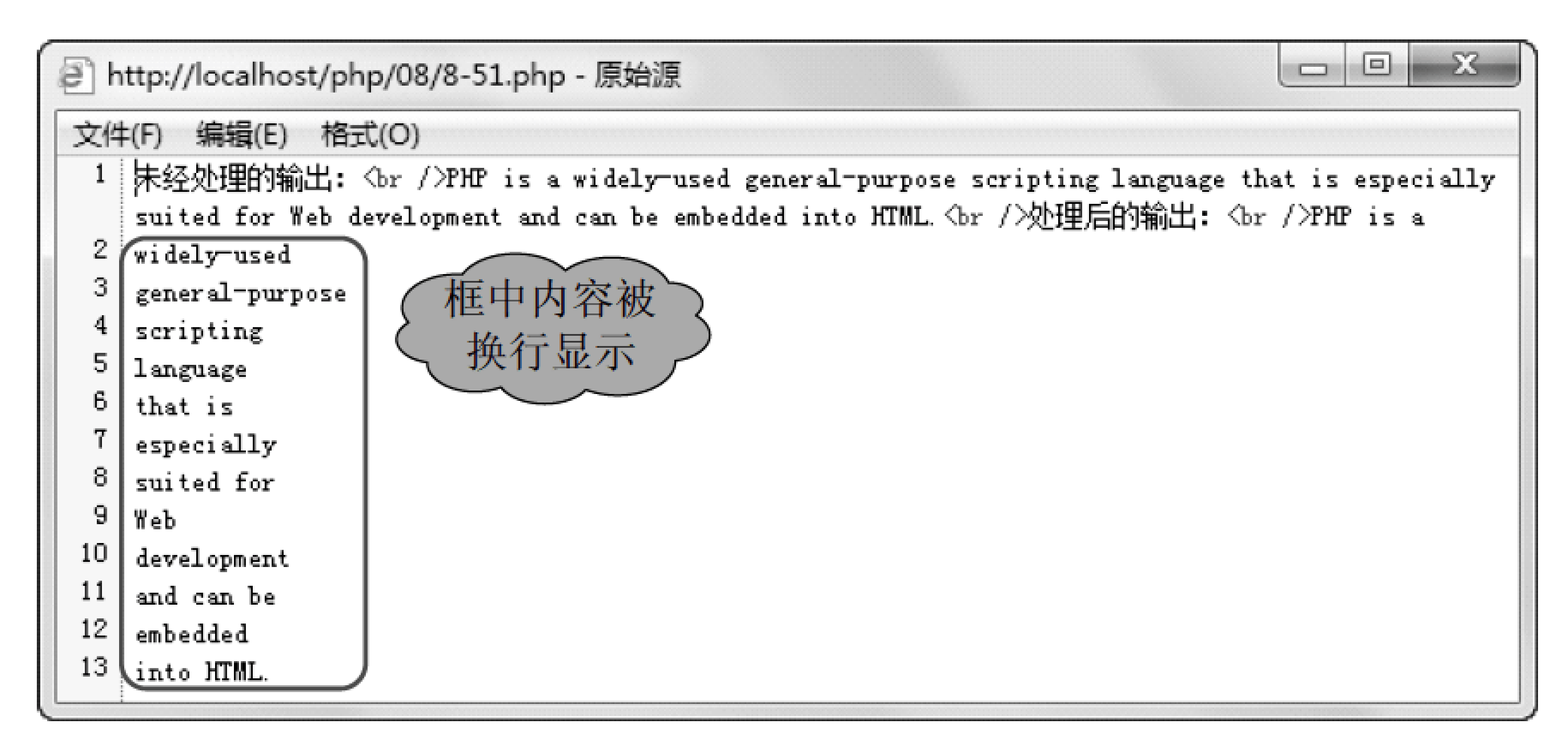


图 8.56　网页源代码

从源代码中可以看到，原来的长字符串被分割为多行显示。需要注意的是，在默认情况下该函数不会截断整个单词，这就使得每行可能多余或者少于指定的字符数。我们接着来看使用 wordwrap()函数的第 3 个可选参数来设置插入的换行符。

【示例 8-52】以下代码演示通过 wordwrap()函数的可选参数 break 来指定插入的换

行符。

```
<?php
    $str='PHP is a widely-used general-purpose scripting language that
    is especially suited for Web development and can be embedded into
    HTML.';                                        //定义一个长的字符串
    echo "未经处理的输出：<br />{$str}";          //输出未经处理的原字符串
    $str=wordwrap($str,15,'<br />'); //设置最大的截断宽度为 15 并设置换行符
    echo "<br />处理后的输出：<br />{$str}";  //输出处理后的字符串
?>
```

代码运行结果如图 8.57 所示。



图 8.57　运行结果

在将默认的换行符设置为浏览器可以识别的“
”符号后，我们就可以从浏览器中直接看到处理后的效果。

8.6　比较字符串

比较字符串是在编程过程中常用的操作。虽然 PHP 为我们提供了比较运算符可以用来比较字符串，但是其功能极其有限。本章我们就来介绍一些常用的比较字符串函数。

8.6.1　strcmp()和 strcasecmp()函数

strcmp()函数使用二进制安全地比较两个字符串，strcasecmp()函数与 strcmp()函数的不同就在于 strcasecmp()函数在比较的过程中不区分字母大小写。

```
int strcmp ( string $str1 , string $str2 )
int strcasecmp ( string $str1 , string $str2 )
```

参数 str1 为比较的第一个字符串；参数 str2 为比较的第二个字符串。该函数通常会返回一个数字来代表比较的结果：

- 返回小于 0 的数值：str1 小于 str2。
- 返回 0：str1 等于 str2。
- 返回大于 0 的数值：str1 大于 str2。

比较数组的规则比较简单，这里就不详细介绍。该函数在比较字母的时候会根据字母的 ASCII 码进行比较，这里就不列出详细的 ASCII 码表了，我们只写出字母和数字的 ASCII 码值的范围：

- A～Z 的 ASCII 码值的范围为 65～90；
- a～z 的 ASCII 码值的范围为 97～122；
- 0～9 的 ASCII 码值的范围为 48～57。

【示例 8-53】以下代码演示使用 strcmp()和 strcasecmp()函数比较两个字符的大小。

```
<?php
    //定义两个字符串
    $str1='ABC';
    $str2='abc';
    //使用 strcmp 和 strcasecmp 函数进行比较
    $res1=strcmp($str1,$str2);
    $res2=strcasecmp($str1,$str2);
    echo "{$str1}与{$str2}区分大小写比较结果为{$res1}";
    echo "<br />{$str1}与{$str2}不区分大小写比较结果为{$res2}";
?>
```

代码运行结果如图 8.58 所示。从运行结果可以得知，区分大小写比较的情况下字符串 ABC 小于字符串 abc；而不区分大小写进行比较的情况下则相等。

当数字和字母进行比较的时候总会返回字母大于数字的结果，因为字母的 ASCII 码值总大于数字的 ACSII 码值。在进行比较字符串的时候，函数会对两个字符串对应的字符串进行比较，如果第一个字符就可以得出结果则比较结束，否则逐个向后比较直到可以得出结果。

【示例 8-54】以下代码演示使用 strcmp()函数比较两个字符串的大小。

```
<?php
    //定义两个字符串
    $str1='ABc';
    $str2='ABC';
    $res=strcmp($str1,$str2);        //使用函数进行比较
    echo "{$str1}和{$str2}的比较结果为{$res}";
?>
```

代码运行结果如图 8.59 所示。



图 8.58　运行结果



图 8.59　运行结果

从运行结果我们可以得知字符串“ABc”要大于字符串“ABC”，这是因为字符串“c”的 ASCII 码大于“C”的 ASCII 码。

8.6.2　strncmp()和 strncasecmp()函数

strncmp()函数与 strncasecmp()函数类似，主要的区别在于 strncmp()函数可以指定比较

字符串的长度；strncasecmp()函数是 strncmp()函数不区分大小写的形式，它们的原型如下：

```
int strncmp ( string $str1 , string $str2 , int $len )
int strncasecmp ( string $str1 , string $str2 , int $len )
```

参数 str1 和 str2 即为需要比较的两个字符串；参数 len 可以指定比较的长度。

【示例 8-55】以下代码演示使用 strncmp()函数比较指定长度字符串的大小。

```
<?php
    //定义两个字符串
    $str1='ABCDEF';
    $str2='ABCDef';
    //比较字符串并输出运行结果
    $res=strncmp($str1,$str2,3);
    echo "{$str1}和{$str2}比较前 3 个字符串的结果为：{$res}";
    $res=strncmp($str1,$str2,5);
    echo "<br />{$str1}和{$str2}比较前 5 个字符串的结果为：{$res}";
?>
```

代码运行结果如图 8.60 所示。



图 8.60　运行结果

从代码中可以看出$str1 和$str2 的值是不相同的，但是前 4 个字符是相同的。因此，当把字符串比较限制为前 3 个字符时会得到相等的结果，而在比较前 5 个字符时会得到不同的结果。strcasecmp()函数的使用比较简单，这里就不再详细介绍。

8.6.3　strnatcmp()和 strnatcasecmp()函数

strnatcmp()函数用于使用自然排序算法比较字符串；strnatcasecmp()函数是 strnatcmp()函数不区分大小写的形式，它们的原型如下：

```
int strnatcmp ( string $str1 , string $str2 )
int strnatcasecmp ( string $str1 , string $str2 )
```

参数 str1 和 str2 即为需要比较的字符串。所谓自然排序法是指符合人类习惯的方式，例如，我们将字符串 str2 和字符串 str11 进行比较的话，会认为字符串 str11 大于 str2。但是使用 strcmp()函数进行比较的话，它会得出字符串 str2 大于字符串 str11 的结果。而 strnatcmp()函数则会得出与我们期望相同的结果。

【示例 8-56】以下代码演示使用 strcmp()函数和 strnatcmp()函数运行结果的区别。

```
<?php
    //定义两个字符串
    $str1='str2';
    $str2='str11';
    //使用两个函数分别比较其大小并输出结果
    $res1=strcmp($str1,$str2);
    $res2=strnatcmp($str1,$str2);
```

```
    echo "使用 strcmp 函数比较{$str1}和{$str2}的结果为{$res1}";
    echo "<br />使用 strnatcmp 函数比较{$str1}和{$str2}的结果为{$res2}";
?>
```

代码运行结果如图 8.61 所示。

从运行结果可以看到，代码输出了我们前面所推断的结果。strnatcasecmp()函数的使用这里就不做详细介绍了。

8.6.4　substr_compare()函数

substr_compare()函数可以说是 strncmp()函数的增强版本，该函数不仅可以指定需要比较的字符串的长度，而且可以指定比较开始的位置，它的原型如下：

```
int substr_compare ( string $main_str , string $str , int $offset [, int
$length [, bool $case_insensitivity = false ]] )
```

参数 main_str 和 str 为需要进行比较的两个字符串；参数 offset 为开始比较的偏移量，如果为负数则从末尾开始；可选参数 length 为比较的长度，默认值为 str 和 main_str 的长度减去位置偏移量 offset 后二者中较大的值；可选参数 case_insensitivity 用来规定比较是否区分大小写，FALSE 为区分大小写。

【示例 8-57】以下代码演示 substr_compare()函数的简单使用方式。

```
<?php
    //定义两个字符串
    $str1='Hello world!';
    $str2='world';
    //比较大小并输出结果
    $res=substr_compare($str1,$str2,0);
    echo "从偏移量为 0 的位置开始比较{$str1}与{$str2}的结果为{$res}";
?>
```

代码运行结果如图 8.62 所示。



图 8.61　运行结果



图 8.62　运行结果

这个示例的运行结果很容易理解，因为字母 H 的 ASCII 码值要小于字母 w 的 ASCII 码值，因此结果为－1。

【示例 8-58】以下代码演示使用 substr_compare()函数设置偏移量后比较两个字符串。

```
<?php
    //定义两个字符串
    $str1='Hello world!';
    $str2='world';
    //比较大小并输出结果
    $res=substr_compare($str1,$str2,6);
    echo "从偏移量为 0 的位置开始比较{$str1}与{$str2}的结果为{$res}";
?>
```

代码运行结果如图 8.63 所示。

从运行结果可以看出，比较的结果为 1，说明字符串$str1 在设置偏移量后大于字符串$str2。由于我们为$str1 设置了偏移量，那么$str1 就相当于“world!”，由于比较的长度默认为取长的一个，因此字符串“world!”会与字符串“world”比较，那么结果就很容易得出为 1 了。接下来我们就自己设置比较的长度后进行比较。

【示例 8-59】以下代码演示使用 substr_compare()函数设置偏移量和比较长度后比较两个字符串的大小。

```
<?php
    //定义两个字符串
    $str1='Hello world!';
    $str2='world';
    //比较大小并输出结果
    $res=substr_compare($str1,$str2,6,5);
    echo "从偏移量为 0 的位置开始比较{$str1}与{$str2}的结果为{$res}";
?>
```

代码运行结果如图 8.64 所示。



图 8.63　运行结果

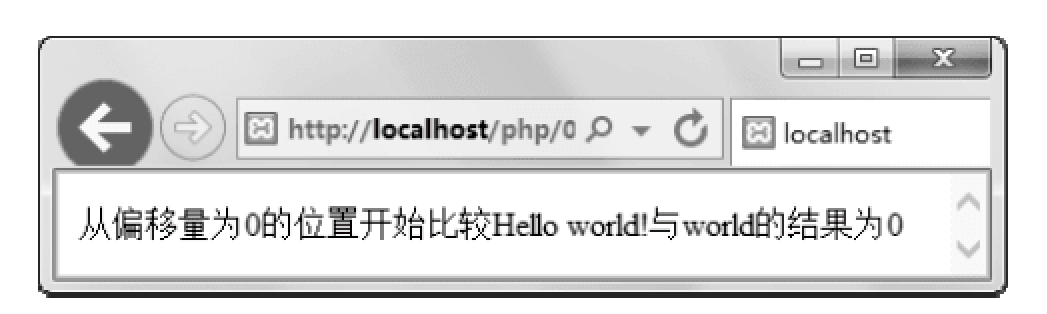


图 8.64　运行结果

该示例中我们指定了比较的长度为 5，虽然$str1 设置偏移量后相当于字符串“world!”，但是我们设置的比较长度为 5，这就导致该字符串实际参与比较的字符串为 world，而字符串$str2 也为 world，那么比较结果理所应当地为 0。

8.7　字符串加密

字符串加密通常用在密码以及安全通行的方面，这里我们介绍两个可以简单进行加密的函数，分别为 md5()函数和 sha1()函数，它们的原型如下：

```
string md5 ( string $str [, bool $raw_output = false ] )
string sha1 ( string $str [, bool $raw_output = false ] )
```

参数 str 即为需要加密的字符串；可选参数 raw_output 用来控制输出的格式，默认为十六进制形式的字符串，如果设置为 TRUE 则返回二进制形式的字符串。MD5 和 SHA1 为两种加密技术的名称。

【示例 8-60】以下代码演示使用 md5()和 sha1()函数加密一段字符串。

```
<?php
    $str="Hello world!";          //定义一个字符串
    //使用两种方法进行加密
    $res1=md5($str);
    $res2=sha1($str);
    echo "使用 MD5 算法将字符串{$str}加密后的结果为：{$res1}";
    echo "<br />使用 sha1 算法将字符串{$str}加密后的结果为：{$res2}";
```

```
08  ?>
```

代码运行结果如图 8.65 所示。



图 8.65　运行结果

这两个函数的使用非常简单，这里就不做过多的介绍。

8.8　小　　结

本章中主要介绍了 PHP 中字符串处理的相关函数。字符串处理是 PHP 技术的核心部分，因此提供了类型多样的处理函数。本章中只是讲解了常用的一部分，但是对每种类型的每个参数用法及含义都做了比较详细的介绍。真正掌握了这些知识后，读者在以后即使使用一个新的函数也并不会感到困难。

8.9　本 章 习 题

1. 将 M_PI 预定义常量格式化输出为保留两位小数的浮点数。
2. 去除以下 HTML 代码中的标签并输出。

```
<a href=#><b>下一页</b></a>
```

3. 将以下字符串全部转换为大写并输出。

```
Happy birthday.
```

4. 将以下字符串中的 banana 替换为 pear 并输出。

```
This is a banana.
```

5. 将以下字符串中的单词存储在数组$words 中并输出该数组的详细信息。

```
Nice to meet you
```

6. 将以下字符串使用 MD5 算法加密并输出。

```
This is my password
```

第 9 章　文件系统操作

在 PHP 中，一些数据可以存储在数据库中，而有的数据通常会存储在文件系统中，例如我们前面所学习到的异常记录文件。还有的情况需要从文件系统读取文件，例如图像、配置信息或者是一个 Web 页面。本章就来介绍一些常用的文件系统操作函数。

9.1　目　　录

目录在 Linux 系统中也是一个文件，被称为目录文件，而在 Windows 系统中，我们可以将目录理解为文件夹。例如我们会说，我们写的 PHP 代码会保存在 D:\xampp\htdocs 目录中，我们就可以理解为我们写的 PHP 代码保存在 D 盘中的 xampp 文件夹的 htdocs 子文件夹中。本节中，我们就来介绍一些目录操作的函数。

9.1.1　目录的基础知识

用户在磁盘上寻找文件时，所历经的文件夹线路叫路径。路径有两种使用形式，即绝对路径和相对路径。

1. 绝对路径

例如“D:\xampp\htdocs”就是一个绝对路径，因为它将每层目录都做了明确的说明。我们很容易根据这个目录找到对应的文件，如图 9.1 所示。

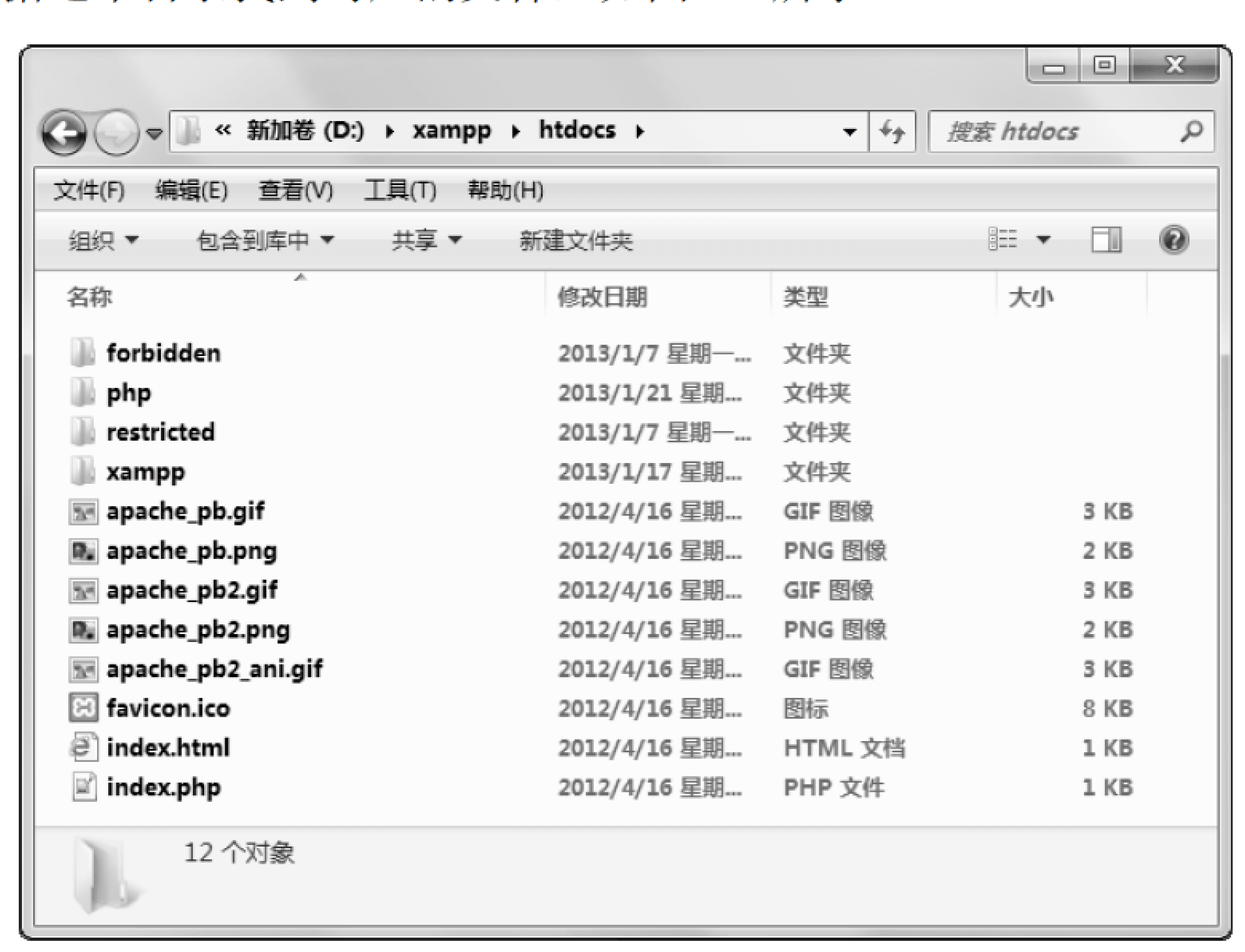


图 9.1　D:\xampp\htdocs

例如图中的 apache_pb.gif 文件，它的绝对路径为“D:\xampp\htdocs\apache_pb.gif”，我们根据这个路径依次经过每个文件夹，就可以找到 apache_pb.gif 这个文件。

2. 相对路径

从字面意义上也比较容易理解相对路径。相对路径不会直接描述一个文件的目录，而是会以一个文件或者目录作为参考，然后以这个参考文件或者目录来描述路径。

在相对路径中，有两个特殊的目录，如下所示。

```
.
..
```

注意：在 Windows 下，路径中通常使用反斜杠“\”来分割目录，但是也可以使用正斜杠“/”。而在 Linux 下为使用正斜杠“/”来分割目录。我们为了统一，将在后面的讲解中使用正斜杠。

“./”目录表示当前所在的目录；“../”目录表示当前所在目录的上一级目录。假设我们现在的目录为“D:\xampp\htdocs\php”，如图 9.2 所示。



图 9.2　D:\xampp\htdocs\php

那么目录“./”就等同于目录“D:\xampp\htdocs\php”。而目录“../”则等同于“D:\xampp\htdocs”即当前目录的上一级目录。

同样地，仍然假设我们当前目录为“D:\xampp\htdocs\php”，那么如何才能找到“D:\xampp\htdocs\apache_pb.gif”这个文件呢？首先，该目录在当前目录的上一级，因此我们使用“../”将当前目录定位到上一级，现在的部分路径如下：

```
../
```

apache_pb.gif 文件就在该目录下，我们就可以直接加入文件名，如下所示。

```
../apache_pb.gif
```

在当前情况下，“../apache_pb.gif”和“D:\xampp\htdocs\apache_pb.gif”是等价的。我们再来看一个比较复杂的情况。我们现在要基于现在的目录“D:\xampp\htdocs\php”找到“D:\xampp\php\php.ini”文件，如图 9.3 所示。



图 9.3　D:\xampp\php\php.ini

为了阅读方便，这里将现在的所在目录以及目标位置列出来：

```
当前目录：D:\xampp\htdocs\php
目标文件：D:\xampp\php\php.ini
```

首先我们可以看到，这两个目录的交汇目录在“D:/xampp”。因此，需要将当前目录向上返回两级，现在可以写成的相对路径如下：

```
../../
```

该目录就相当于“D:\xampp”。接着需要进入到 php 子文件夹，现在的相对路径可以写成如下：

```
../../php
```

该路径就相当于“D:\xampp\php”。下面就可以找到该文件夹下的 php.ini 文件了。完整的相对路径如下：

```
../../php/php.ini
```

我们使用比较多的篇幅来介绍相对路径是有其必要性的，相对路径相比绝对路径来说，它所受的约束比较少。例如一个项目可以是一个文件夹，而这个文件夹下通常会有很多的子文件夹用来存放一些资源。而如果我们在编程过程中的路径都使用绝对路径，那么当我们将这个项目移动位置后，就需要改动代码中的每处绝对路径。而使用相对路径，因为其中的路径都会相对这个项目文件夹来访问其中的子文件夹，那么通常是不需要进行过多改动便可运行的。

9.1.2　判断文件的属性

由于 PHP 是以 UNIX 文件系统为模型的，因此，会将所有的软件和硬件都视为文件。在对一个文件进行操作时，我们通常需要首先确定该文件的类型，然后再执行相应的操作。本节我们将主要介绍 is_dir()、is_file()和 filetype()函数。

1. 判断文件名是否为目录

在对目录进行操作前，首先需要判断指定的文件是否为目录。使用的函数为 is_dir()

函数，它的原型如下：

```
bool is_dir ( string $filename )
```

参数 filename 即为需要判断的文件（目录也是文件）；如果文件存在并且为目录则返回 TRUE，否则返回 FALSE。下面我们就使用 is_dir()函数来判断我们熟悉的“D:\xampp”是否为目录。

【示例 9-1】以下代码演示使用 is_dir()函数判断“D:\xampp”是否为目录。

```
<?php
    $dir=is_dir('D:/xampp');            //判断参数是否为目录
    echo "判断 D:/xampp 是否为目录：";
    var_dump($dir);                     //输出结果类型
?>
```

代码运行结果如图 9.4 所示。从运行结果中可以得知，is_dir()函数返回了布尔值 true，就说明该文件是一个目录。下面我们再演示为 is_dir()函数传入一个不存在的文件后的执行结果。

【示例 9-2】以下代码演示将一个不存在的文件名作为参数，传递给 is_dir()函数。

```
<?php
    $dir=is_dir('D:/notdir');           //判断参数是否为目录
    echo "判断文件是否为目录：";
    var_dump($dir);                     //输出结果类型
?>
```

代码运行结果如图 9.5 所示。从运行结果可以得知，is_dir()函数返回了布尔值 false，这就说明该文件不是一个目录。下面再来看将一个存在文件的路径作为参数传递给 is_dir()函数。

【示例 9-3】以下代码演示将一个存在的文件名作为参数传递给 is_dir()函数。

```
<?php
    $dir=is_dir('D:/xampp/php/php.ini');          //判断参数是否为目录
    echo "判断文件是否为目录：";
    var_dump($dir);                               //输出结果类型
?>
```

代码运行结果如图 9.6 所示。

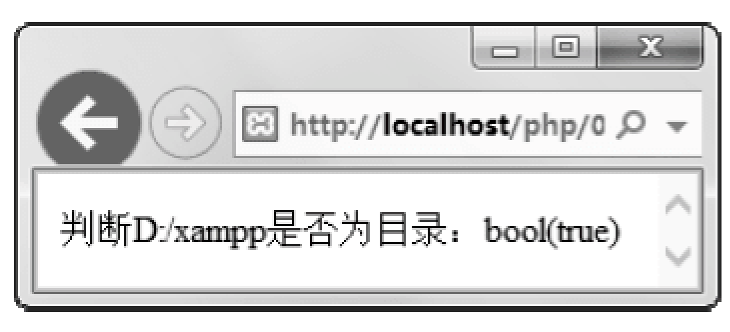


图 9.4　运行结果



图 9.5　运行结果



图 9.6　运行结果

从运行结果可以看出，is_dir()函数在执行后返回了布尔值 false，这就说明该文件不是一个目录。

2. 判断文件名是否为普通文件

与判断文件是否为目录类似，判断文件是否为普通文件使用的函数为 is_file()，该函数的原型如下：

```
bool is_file ( string $filename )
```

参数 filename 即为需要判断的文件名。如果该文件名表示的文件为一个普通的文件，则返回 TRUE，否则返回 FALSE。

【示例 9-4】以下代码演示使用 is_file()函数判断一个文件名是否为一个普通文件。

```
<?php
    $file=is_file('D:/xampp/php/php.ini');
                                        //使用函数判断文件名是否为普通文件
    //根据结果输出不同的信息
    if($file)
        echo "这是一个普通文件。";
    else
        echo "这是不是一个普通文件或者不存在。";
?>
```

代码运行结果如图 9.7 所示。接着我们将一个不存在的文件名作为参数传递给 is_file()函数，来看看执行后的结果。

【示例 9-5】以下代码演示将不存在的文件名作为 is_file()函数的参数。

```
<?php
    $file=is_file('D:/xampp/php/notfile.txt');
                                        //使用函数判断文件名是否为普通文件
    //根据结果输出不同的信息
    if($file)
        echo "这是一个普通文件。";
    else
        echo "这是不是一个普通文件或者不存在。";
?>
```

代码运行结果如图 9.8 所示。



图 9.7　运行结果



图 9.8　运行结果

从运行结果可以得知，该文件不存在或者不为普通文件。该函数的使用和理解都比较简单，这里不再详细介绍。

3. 使用 filetype()函数获取文件类型

filetype()函数不同于 is_dir()和 is_file()函数。该函数用来获取文件的类型而不是判断文件属于何种类型，它的原型如下：

```
string filetype ( string $filename )
```

参数 filename 即为需要判断的文件名。该函数执行成功会返回以下字符串中的一种：

- char：字符串设备；
- block：块设备文件；
- dir：目录类型；

- fifo：命名管道；
- file：普通文件；
- link：符号链接；
- unknown：未知文件类型。

注意：在 Windows 系统下只会返回 file、dir 和 unknown 这 3 种文件类型。

如果函数出错，则返回 FALSE，例如文件不存在的情况。

【示例 9-6】以下代码演示使用 filetype()函数试图获取一个不存在的文件类型。

```
<?php
    $file=filetype('D:/xampp/php/notfile.txt');          //获取文件类型
    var_dump($file);
?>
```

代码运行结果如图 9.9 所示。



图 9.9　运行结果

从运行结果可以看到，虽然 filetype()函数返回了 FALSE，但是也输出了一条警告信息。为了避免输出这条错误信息，我们可以在使用 filetype()函数获取文件类型之前首先判断指定的文件是否存在。file_exists()函数可以用来判断一个文件或者目录是否存在，它的原型如下：

```
bool file_exists ( string $filename )
```

参数 filename 即为需要判断文件名。如果 filename 指定的文件或者目录存在，则函数返回 TRUE，否则返回 FALSE。接下来我们就可以改进示例 9-6 中的代码，使其更完善。

【示例 9-7】以下代码演示使用 file_exists()函数在 filetype()函数使用前判断文件是否存在。

```
<?php
    $filename='D:/xampp/php/notfile.txt';   //判断文件是否存在
    if(file_exists($filename))
        echo "该文件的类型为：".filetype($filename);
                                            //如果文件存在则输出文件类型
    else
        echo '该文件或者目录不存在。';         //不存在则输出提示
?>
```

代码运行结果如图 9.10 所示。由于该文件确实不存在于硬盘中，因此程序运行后会提示相关信息。将一个存在的文件作为参数传递后，该代码会输出文件的类型。

【示例 9-8】以下代码演示获取一个存在的文件类型。

```
<?php
    $filename='D:/xampp/php/php.ini';       //判断文件是否存在
    if(file_exists($filename))
```

```
        echo "该文件的类型为：".filetype($filename);
                                              //如果文件存在则输出文件类型
    else
        echo '该文件或者目录不存在。';             //不存在则输出提示
?>
```

代码运行结果如图 9.11 所示。从运行结果中可以看出，该文件为普通文件。接下来我们将一个目录传递给 filetype()函数来进行判断。

【示例 9-9】以下代码演示使用 filetype()函数获取文件类型。

```
<?php
    $filename='D:/xampp/php';                 //判断文件是否存在
    if(file_exists($filename))
        echo "该文件的类型为：".filetype($filename);
                                              //如果文件存在则输出文件类型
    else
        echo '该文件或者目录不存在。';             //不存在则输出提示
?>
```

代码运行结果如图 9.12 所示。



图 9.10　运行结果



图 9.11　运行结果



图 9.12　运行结果

从运行结果可以得知，该文件为目录。

9.1.3　获取文件信息

除了获取文件的类型之外，PHP 还提供了多个函数来获取文件的信息，例如文件的大小、修改时间和访问时间等信息。

1. 获取文件访问和修改时间

获取文件访问和修改时间使用的函数是 fileatime()函数和 filemtime()函数，它们的原型如下：

```
int fileatime ( string $filename )
int filemtime ( string $filename )
```

参数 filename 即为需要获取访问和修改时间的文件名。如果函数执行成功，则返回 UNIX 时间戳形式的时间，如执行失败则返回 FALSE。UNIX 时间戳可以使用 date()函数进行格式化输出。

【示例 9-10】以下代码演示使用 fileatime()函数和 filemtime()函数获取文件的访问和修改时间，并使用 date()函数格式化时间。

```
<?php
    $filename='D:/xampp/php/php.ini';             //定义文件名
    if(file_exists($filename)){                   //判断文件是否存在
```

```
        if(filetype($filename)=='file'){        //判断文件是否为普通文件
            //获取文件的相关信息并格式化输出
            echo '该文件最后访问的时间为：'
                        .date('Y-m-d',fileatime($filename));
            echo '<br />该文件最后修改的时间为：'
                        .date('Y-m-d',filemtime($filename));
        }
    }
?>
```

以上代码运行结果如图 9.13 所示。程序运行后就获取到了该文件的相关信息，当然我们可以通过查看该文件的属性来验证程序获取到的信息的正确性。该文件的属性如图 9.14 所示。



图 9.13　运行结果　　　　图 9.14　php.ini 文件属性

从图 9.14 所示的文件属性中可以得知，程序获取到的结果是正确的。

2. 获取文件大小

获取文件大小可以使用 filesize()函数，它的原型如下：

```
int filesize ( string $filename )
```

参数 filename 即为需要获取大小的文件名。该函数执行成功后会返回该文件的字节数，如果出错则返回 FALSE 并生成一条 E_WARNING 级的错误。

【示例 9-11】以下代码演示使用 filesize()函数获取文件大小。

```
<?php
    $filename='D:/xampp/php/php.ini';       //定义文件名
    if(is_file($filename))                  //判断文件是否存在
        echo '该文件的大小为'.filesize($filename).'字节。';
                                            //获取文件的大小并输出
?>
```

以上代码运行结果如图 9.15 所示。我们可以看到，程序运行后成功获取到了文件的大小。而在平常的使用中我们更喜欢使用兆字节作为单位，因此可以在输出时进行单位转换。

当然，filesize()函数也可以用来获取目录文件的大小。

【示例 9-12】以下代码演示使用 filesize()函数获取文件的大小，并以兆字节为单位的形式输出。

```
<?php
    $filename='./file.txt';                //定义文件名
    if(is_file($filename))
    printf('该文件的大小为%dM 字节。',filesize($filename)/1024/1024);
                                           //获取文件的大小并格式化输出
?>
```

代码运行结果如图 9.16 所示。

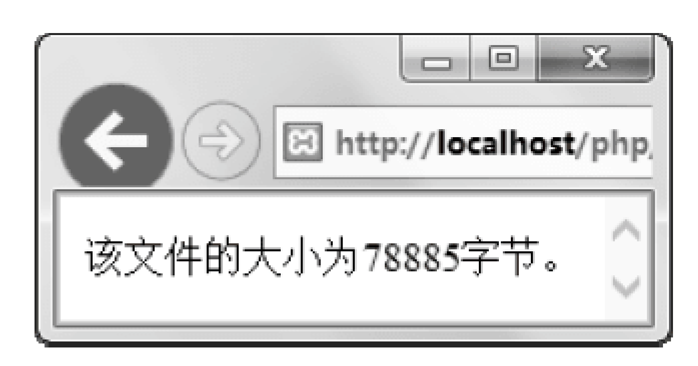


图 9.15　运行结果



图 9.16　运行结果

从运行结果可以看到这种显示更加易读一些。

3. 使用 stat()函数获取文件信息

stat()函数可以获取到文件的多个信息并以数组的形式返回，该函数的原型如下：

```
array stat ( string $filename )
```

参数 filename 即为需要获取信息的文件。该函数返回的数组信息如表 9.1 所示。

表 9.1　stat()函数返回的数组信息

数字下标	关联键名	说　　明
0	dev	设备名
1	ino	inode 码
2	mode	inode 保护模式
3	nlink	被连接数目
4	uid	所有者的用户 id
5	gid	所有者的组 id
6	rdev	设备类型
7	size	文件大小的字节数
8	atime	上次访问时间（UNIX 时间戳形式）
9	mtime	上次内容修改时间（UNIX 时间戳形式）
10	ctime	上次属性改变时间（UNIX 时间戳形式）
11	blksize	文件系统 I/O 的块大小，Windows 下会返回－1
12	blocks	所占据块的数目，Windows 下会返回－1

注意：修改时间是指文件内容被修改的时间；改变时间是指文件的权限被改变的时间，例如可执行权限等。

【示例 9-13】以下代码演示使用 stat()函数获取一个文件的信息。

```
<?php
    $state=stat('D:/xampp/php/php.ini');        //获取文件信息
    //输出文件的相关信息
    printf('该文件的大小为%dKb',$state['size']/1024);
    echo '<br />该文件的最后访问时间是：'.date('Y-m-d',$state['atime']);
    echo '<br />该文件的最后修改时间是：'.date('Y-m-d',$state['mtime']);
    echo '<br />该文件的最后改变时间是：'.date('Y-m-d',$state['ctime']);
?>
```

代码运行结果如图 9.17 所示。从运行结果可以看到，该函数可以比较简便地获取到文件的一些信息，我们在实际应用中可以根据情况选择 stat()函数或者前面介绍的函数。

4. 文件权限操作

文件的权限在 Linux 系统中尤为重要，而在 Windows 系统就比较弱化了。例如在 Linux 中的文件都会有读写权限和执行权限的限制。PHP 中提供了多个函数来判断文件是否允许指定的操作。我们首先来看这些函数的原型：

```
bool is_executable ( string $filename )
bool is_readable ( string $filename )
bool is_writable ( string $filename )
```

这 3 个函数的 filename 参数均为需要进行判断的文件名。is_executable 函数用来判断文件是否可执行；is_readable()函数用来判断文件是否可以读；is_writable()函数用来判断文件是否可写。

【示例 9-14】以下代码演示判断一个文件是否允许执行指定的操作。

```
<?php
    $filename='./xampp.exe';
    if(is_readable($filename))
        echo '该文件允许读操作<br />';
    else
        echo '该文件不允许读操作<br />';
    if(is_writable($filename))
        echo '该文件允许写操作<br />';
    else
        echo '该文件不允许写操作<br />';
    if(is_executable($filename))
        echo '该文件允许执行';
    else
        echo '该文件不允许执行';
?>
```

代码运行结果如图 9.18 所示。

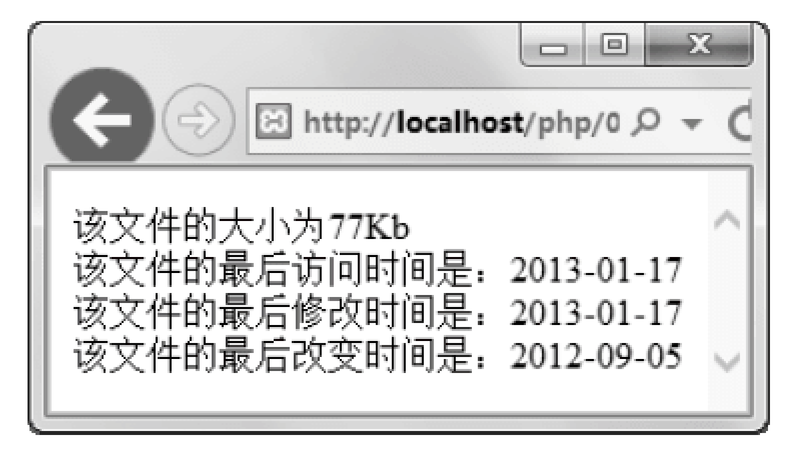


图 9.17　运行结果



图 9.18　运行结果

5. 获取文件系统或磁盘分区空间信息

在文件操作的时候，需要确定目标位置是否有足够的空间来存放该文件。PHP 提供了两个函数来获取文件系统或者磁盘分区的空间信息。它们的函数原型如下：

```
float disk_free_space ( string $directory )
float disk_total_space ( string $directory )
```

参数 directory 即为文件系统或者磁盘分区名称。disk_free_space()函数会返回文件系统或者磁盘分区可用的字节数；disk_total_space()函数会返回文件系统或者磁盘分区总共的字节数。

【示例 9-15】以下代码演示使用 disk_free_space()函数和 disk_total_space()函数，获取磁盘分区和文件系统可用的及总共的字节数。

```
<?php
    //获取磁盘空间信息并格式化后输出
    printf('C 盘的总容量为：%.2fGb。',disk_total_space('C:')/1024/1024/
        1024);
    printf('<br />D 盘的可用容量为：%.2fGb。',disk_free_space('D:')/1024/
        1024/1024);
?>
```

代码运行结果如图 9.19 所示。



图 9.19　运行结果

我们可以通过磁盘管理来验证获取到的信息，如图 9.20 所示。



图 9.20　查看磁盘信息

从运行结果和实际的磁盘信息对比可知，使用 disk_free_space()和 disk_total_space()函

数可以准确获取到磁盘的可用空间和总空间。那么我们当然可以获取到磁盘已经使用的空间，只需做一个减法即可，这里就不再做演示。

9.1.4　目录操作

在文件系统操作中，对目录的操作是必不可少的。本节中我们就来介绍一些目录操作的函数。

1. 获取当前工作目录

获取当前工作目录可以使用 getcwd()函数，它的原型如下：

```
string getcwd ( void )
```

该函数不接收任何参数。

【示例 9-16】以下代码演示使用 getcwd()函数获取当前工作目录。

```
<?php
    echo '当前工作目录为：'.getcwd();
?>
```

代码运行结果如图 9.21 所示。



图 9.21　运行结果

我们可以看到代码在运行后输出了当前工作目录。

2. 改变工作目录

该变工作目录可以使用 chdir()函数，它的原型如下：

```
bool chdir ( string $directory )
```

参数 directory 即为将要切换到的新目录。为了便于大家理解，我们首先在下面的示例源代码同级目录下建立一个 testdir 文件夹，如图 9.22 所示。

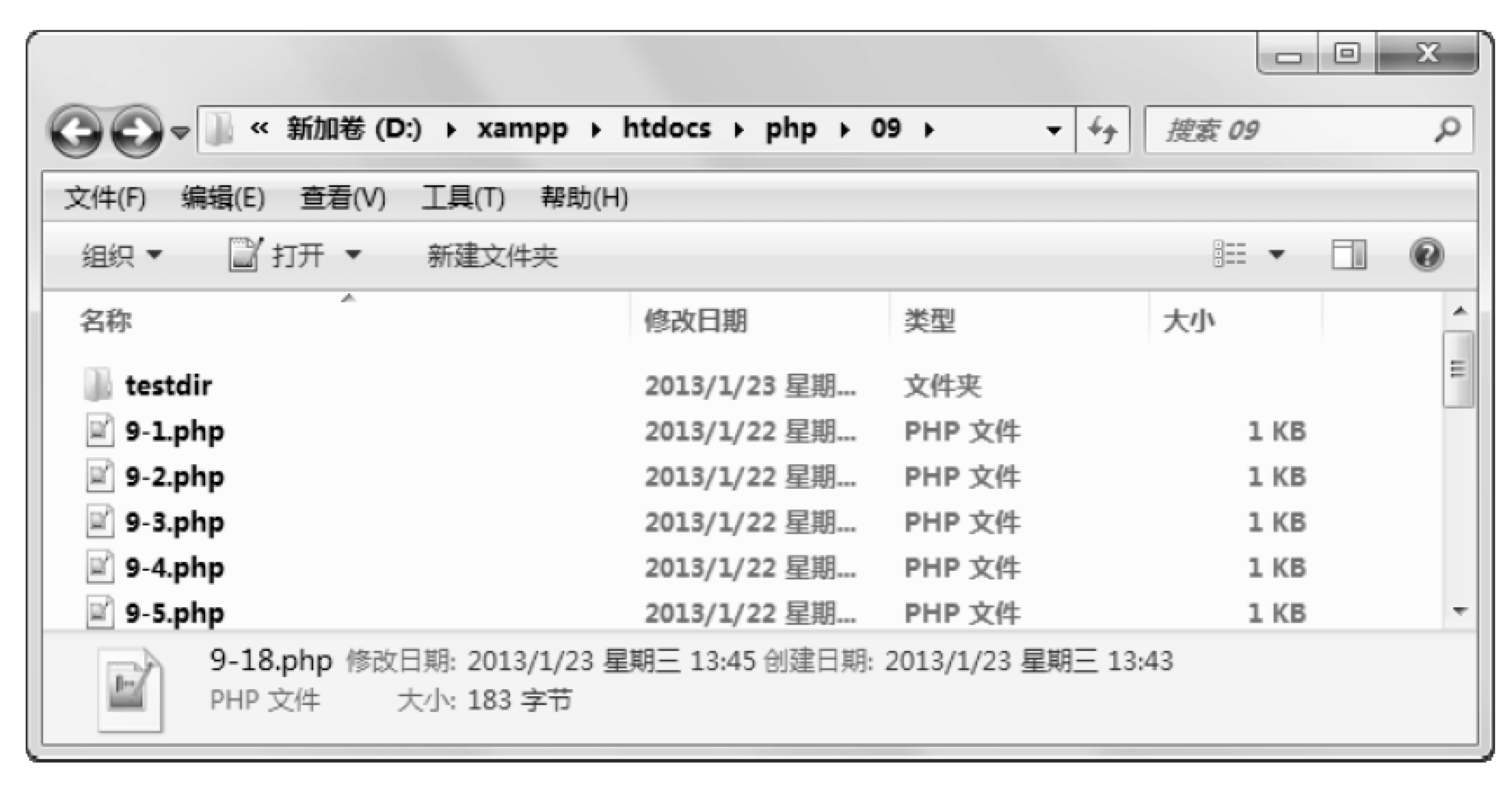


图 9.22　演示前的准备

【示例 9-17】以下代码演示使用 chdir()函数改变当前的工作目录。

```
<?php
    echo '改变工作目录前的工作目录为：'.getcwd();
    chdir('./testdir');                                    //改变工作目录
```

```
    echo '<br />改变工作目录后的工作目录为：'.getcwd();
?>
```

代码运行结果如图 9.23 所示。

从运行结果可以看出，当前工作目录在使用 chdir()函数更改后已经从“D:\xampp\htdocs\php\09”变为“D:\xampp\htdocs\php\09\testdir”。当然，我们也可以绝对路径作为参数，这里就不再详细演示。

3. 创建目录

创建目录可以使用 mkdir()函数，它的原型如下：

```
bool mkdir ( string $pathname [, int $mode [, bool $recursive [, resource
$context ]]] )
```

参数 pathname 为需要创建的目录名称；可选参数 mode 用来规定文件夹的访问权限，该选项会在 Windows 系统中被忽略；可选参数 recursive 规定是否可以递归地创建目录；可选参数 context 用来规定文件句柄的环境。为了方便理解，我们将在前面创建的 testdir 文件夹中演示创建文件夹。

【示例 9-18】以下代码演示使用 mkdir()函数在 testdir 文件夹中创建名为 dir 的文件夹。

```
<?php
    chdir('./testdir');      //改变工作目录
    if(mkdir('dir'))         //创建文件夹并输出提示信息
        echo '文件夹创建成功！';
    else
        echo '文件夹创建失败！';
?>
```

代码运行结果如图 9.24 所示。代码运行后提示文件夹创建成功，我们可以通过查看 testdir 文件夹来验证，如图 9.25 所示。



图 9.23　运行结果



图 9.24　运行结果



图 9.25　示例程序 9-18 创建的文件夹

从图 9.25 中我们可以看到程序成功创建了文件夹。在默认的情况下，mkdir()函数不允

许递归地创建文件夹，例如如下的形式：

```
mkdir('dir1/dir2/dir3')
```

上面的代码在运行后会报错。如果需要递归地创建文件夹，就需要设置 mkdir()函数的第 3 个参数为 TRUE。

【示例 9-19】以下代码演示使用 mkdir()函数递归地创建文件夹。

```
<?php
    chdir('./testdir');                          //首先更改工作目录到测试文件夹
    if(mkdir('dir1/dir2/dir3',777,TRUE))         //递归地创建目录
        echo '目录创建成功！';
    else
        echo '目录创建失败！';
?>
```

代码运行结果如图 9.26 所示。我们同样通过查看目录来验证，如图 9.27 所示。



图 9.26　运行结果



图 9.27　示例程序 9-19 创建的目录

4. 返回指定路径中的文件和目录

返回指定路径中的文件和目录需要使用的函数是 scandir()，它的原型如下：

```
array scandir ( string $directory [, int $sorting_order [, resource
$context ]] )
```

参数 directory 即为需要查看的路径；可选参数 sorting_order 用来控制排序方式，默认为按照字母升序排列，如果设置为 1 则按照字母降序排列；可选参数 context 用来规定文件句柄的环境。由于 scandir()函数会返回一个数组，我们可以使用 foreach 结构来遍历输出。

【示例 9-20】以下代码演示使用 scandir()函数输出当前目录下的文件和目录。

```
<?php
    $arr=scandir(getcwd());        //将当前目录作为 scandir()函数的参数
    echo "当前工作目录下的文件和目录如下：<br />";
    foreach($arr as $v)            //遍历数组
        echo "{$v}<br />";
?>
```

代码运行结果如图 9.28 所示。

在这里需要注意的是目录“.”和“..”也会被输出。Scandir()函数可选参数的用法比较简单，这里就不再介绍。

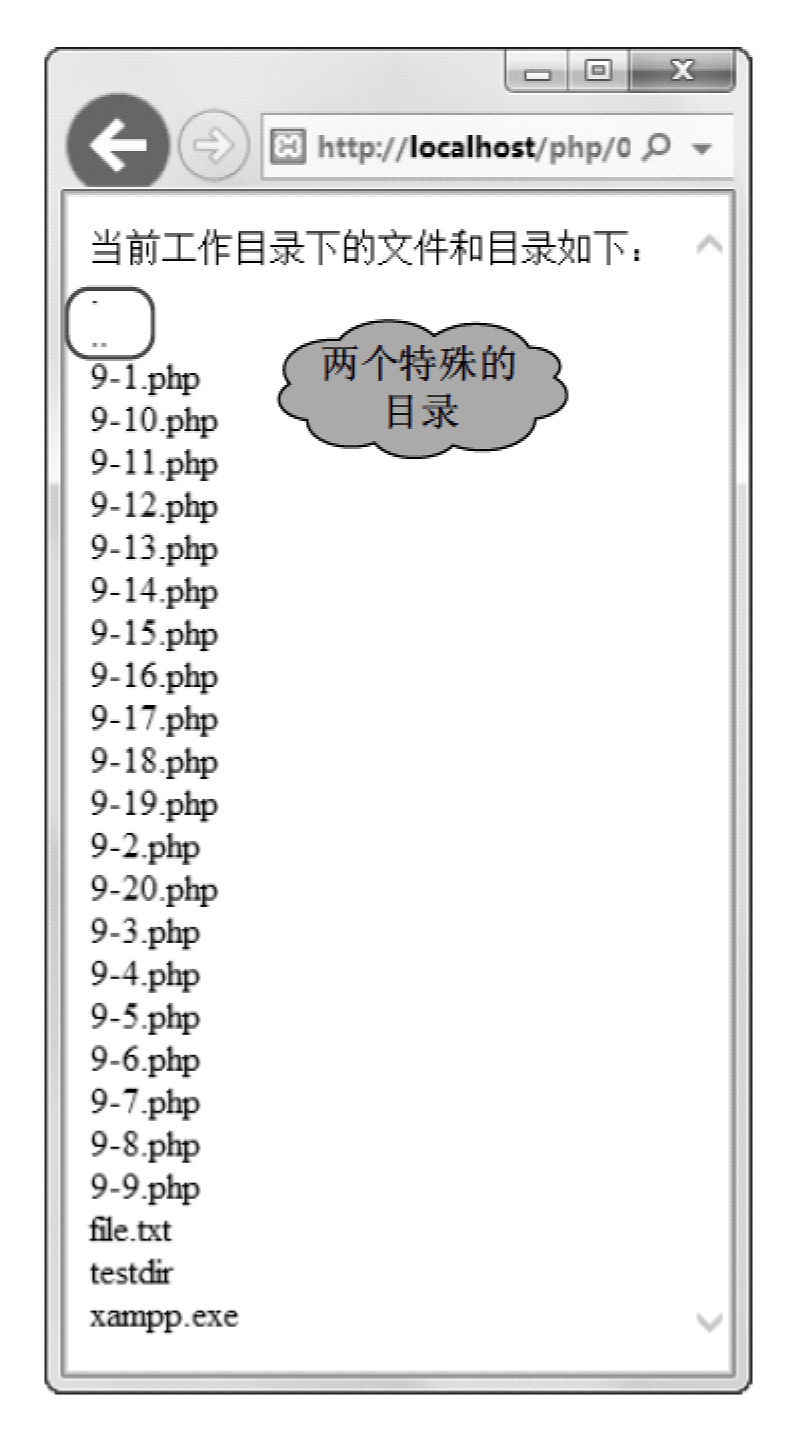


图 9.28　运行结果

5. 利用目录句柄查看目录

利用文件句柄查看目录首先需要打开一个目录句柄，然后再对这个句柄进行操作，操作完成后再关闭目录句柄。打开目录句柄使用到的函数为 opendir()，它的原型如下：

```
resource opendir ( string $path [, resource $context ] )
```

参数 path 为要打开的目录路径；可选参数 context 用来规定文件句柄的环境。操作文件句柄的函数有 readdir()和 rewinddir()函数，它们的原型如下：

```
string readdir ( resource $dir_handle )
void rewinddir ( resource $dir_handle )
```

参数 dir_handle 即为 opendir()函数打开的文件句柄。readdir()函数用来从目录句柄中读取条目；rewinddir()函数用来将目录流倒回到目录开头。关闭目录句柄使用 closedir()函数，它的原型如下：

```
void closedir ( resource $dir_handle )
```

参数 dir_handle 为 opendir()函数打开的文件句柄。下面就来简单演示一下使用打开句柄并获取句柄中的 3 条数据。

【示例 9-21】以下代码演示使用 opendir()函数打开一个目录句柄，并使用 readdir()函数读取其中的 3 条数据。

```
<?php
    $dh=opendir('./');          //将当前目录作为路径打开
    //读取目录句柄中的 3 条数据
    echo '从目录句柄中读取的第一条记录为：'.readdir($dh);
    echo '<br />从目录句柄中读取的第二条记录为：'.readdir($dh);
```

```
    echo '<br />从目录句柄中读取的第三条记录为：'.readdir($dh);
    closedir($dh);                    //关闭打开的目录句柄
?>
```

代码运行结果如图 9.29 所示。从运行结果可以看到，这段程序读取到了 3 条记录。而我通过查看代码可以发现，如果需要遍历整个目录，则完全可以使用循环来更好地解决。

【示例 9-22】以下代码演示使用循环遍历目录。

```
<?php
    $dh=opendir('./');                              //打开目录句柄
    while(($files=readdir($dh))!==FALSE)            //对返回结果进行判断并输出
        echo "{$files}<br />";
    closedir($dh);                                  //关闭目录句柄
?>
```

代码运行结果如图 9.30 所示。



图 9.29　运行结果

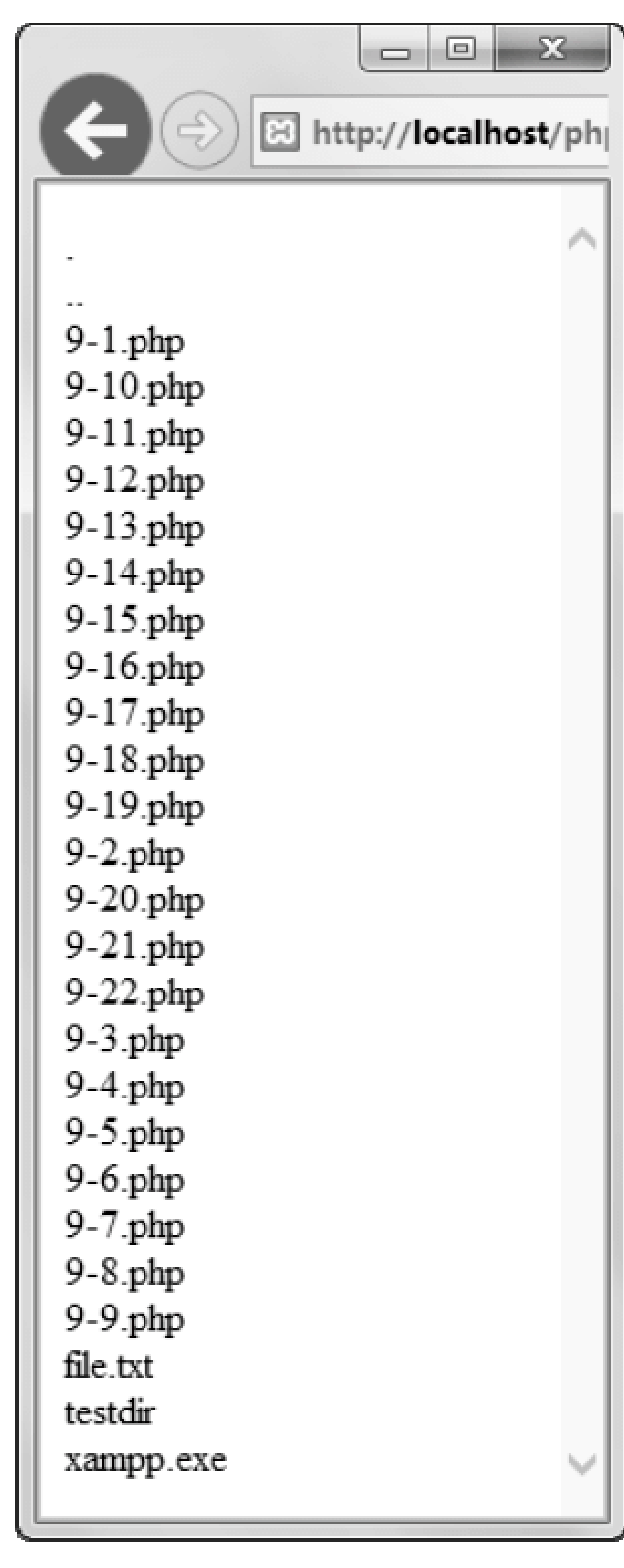


图 9.30　运行结果

在将 readdir()函数的返回值作为判断条件的时候，一定要使用严格相等或不等，因为返回的文件或者目录名称可能为字符串 TRUE 或者 FALSE。如果不使用严格相等或不等，则有可能使遍历过程意外终止。

rewinddir()函数的使用非常简单，我们通过一个简单的示例来介绍。

【示例 9-23】以下代码演示使用 rewinddir()函数重置目录句柄。

```
<?php
    $dh=opendir('./');                //将当前目录作为路径打开
    //读取目录句柄中的 3 条数据
    echo '从目录句柄中读取的第一条记录为：'.readdir($dh);
```

```
05      echo '<br />从目录句柄中读取的第二条记录为：'.readdir($dh);
06      rewinddir($dh);              //重置目录句柄
07      echo '<br />使用 rewinddir 重置目录句柄后，读取到的记录为：'.readdir($dh);
08      closedir($dh);               //关闭打开的目录句柄
09  ?>
```

代码运行结果如图 9.31 所示。

从运行结果可以看出，在代码第 6 行调用 rewinddir()函数后，在代码第 7 行再次调用 readdir()函数会取得该目录的第一个文件。

6. 删除目录

删除目录可以使用 rmdir()函数，它的原型如下：

```
bool rmdir ( string $dirname )
```

参数 dirname 即为需要删除的目录名称。需要注意的是，要删除的目录必须为空目录而且有相应的操作权限。因此，在删除目录前首先需要判断这个目录是否为空，如为空则直接删除该目录，如不为空则首先删除文件夹内的文件。这里我们只简单演示一个删除空目录的示例。

【示例 9-24】以下代码演示使用 rmdir 函数删除一个空目录。

```
<?php
    $path='./testdir/dir';
    if(is_dir($path)){              //判断文件是否为目录
        if(rmdir($path))            //删除目录并输出信息
            echo '目录删除成功！';
        else
            echo '目录删除失败！';
    }
?>
```

代码运行结果如图 9.32 所示。



图 9.31 运行结果



图 9.32 运行结果

删除一个空目录非常简单，这里不多做介绍。删除非空文件夹的知识在我们讲解了文件操作的知识后读者便可掌握。

9.2 简单读取和输出文件

与获取目录信息类似，PHP 提供了多个函数用来简单输出文件中的内容。当然也有同目录操作类似的使用文件句柄来操作文件。我们首先来介绍简单读取文件的函数。

9.2.1　将文件读取到数组

为了方便知识的讲解，我们首先在示例源文件所在文件夹下建立一个文本文件，文本的内容如图 9.33 所示。将文件读取到数组可以使用 file()函数，该函数的原型如下：

```
array file ( string $filename [, int $use_include_path [, resource $context ]] )
```

参数 filename 即为需要读取的文件名；可选参数 use_include_path 用来规定是否在 include_path 中搜寻文件，该选项可以在 PHP 配置文件中设置；可选参数 context 用来规定文件流的环境。该函数会将文件中的一行（包括换行符）作为数组的一个元素。

【示例 9-25】以下代码演示使用 file()函数将文件内容读取到一个数组中。

```
<?php
    $lines=file('file.txt');          //设置文件名
    print_r($lines);                  //输出数组的详细信息
?>
```

代码运行结果如图 9.34 所示。

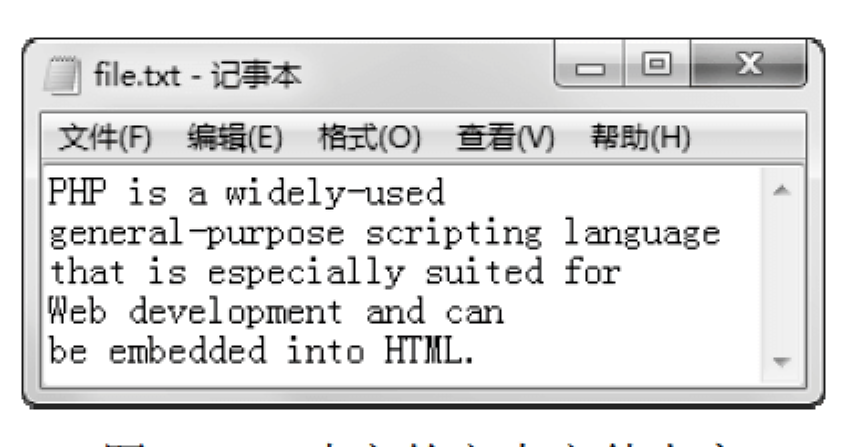


图 9.33　建立的文本文件内容



图 9.34　运行结果

从运行结果可以看到该函数包含了 file.txt 文件中的所有内容。file()函数不仅可以用来读取本地文件的内容，而且可以读取一个 URL 地址的内容。下面就演示使用 file()函数来读取百度搜索的首页面。

【示例 9-26】以下代码演示使用 file()函数读取百度搜索的首页面。

```
<?php
    $lines=file('http://www.baidu.com');          //将百度地址作为参数
    echo $lines[9];                               //输出数组的第 9 个元素
?>
```

代码运行结果如图 9.35 所示。



图 9.35　运行结果

从运行结果可以看到输出该页面源代码的第 9 行为百度的标志。当然，也可以通过遍

历来输出整个页面，这里就不再详细演示。

9.2.2　将文件读取到字符串

将文件读取到字符串可以使用函数 file_get_contents()，它的原型如下：

```
string file_get_contents ( string $filename [, bool $use_include_path [,
resource $context [, int $offset [, int $maxlen ]]]] )
```

参数 filename 即为需要读取的文件名称；可选参数 use_include_path 用来指定是否从 PHP 配置文件中指定的位置读取文件，如果需要读取则使用 FILE_USE_INCLUDE_PATH 常量；可选参数 context 用来规定文件流的选项；可选参数 offset 用来规定读取文件的偏移量，即指定开始读取的位置；可选参数 maxlen 可以规定读取的最大长度。

【示例 9-27】以下代码演示使用 file_get_contents()函数读取指定文件中的内容到字符串。

```
<?php
    $files=file_get_contents('file.txt');       //指定读取的文件
    echo $files;                                //输出读取到的字符串
?>
```

代码运行结果如图 9.36 所示。从运行结果可以看到，file_get_contents()函数读取出了我们建立的文件的所有内容。下面就使用 offset 和 maxlen 参数来限制读取。

【示例 9-28】以下代码演示使用 offset 和 maxlen 参数限制 file_get_contents()函数读取文件。

```
<?php
    $files=file_get_contents('file.txt',FALSE,NULL,15,15);
                                                //限制读取方式
    echo $files;                                //输出读取到的字符串
?>
```

代码运行结果如图 9.37 所示。

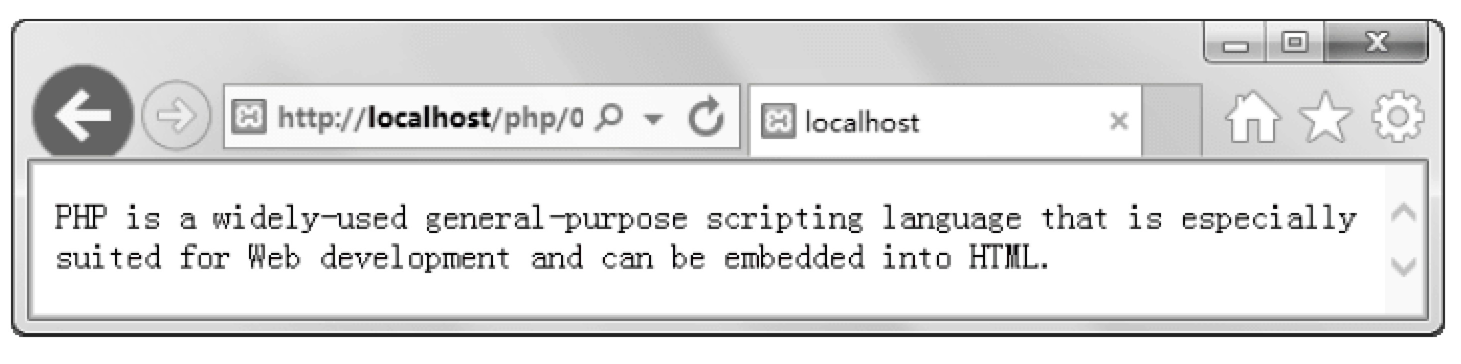


图 9.36　运行结果

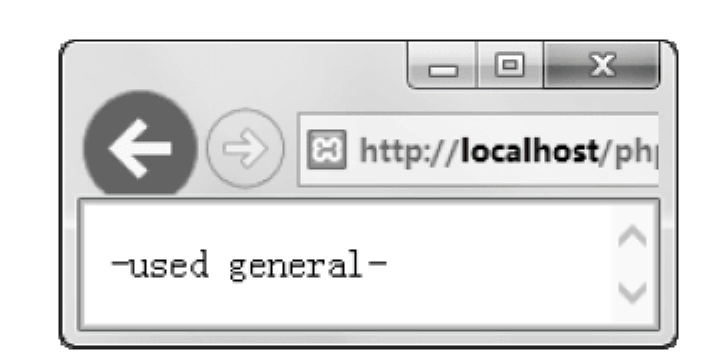


图 9.37　运行结果

当然，file_get_contents()函数也可以用来读取一个 URL，这里我们不做详细介绍。

9.2.3　将文件直接输出

将文件直接输出可以使用 readfile()函数，相对于前面介绍的函数来说，该函数的执行效果更加直接，它的原型如下：

```
int readfile ( string $filename [, bool $use_include_path [, resource
$context ]] )
```

参数 filename 为需要输出的文件名；可选参数 use_include_path 可以设置是否从 PHP 配置文件指定的位置读取文件；context 为文件流的控制参数。该函数正确执行后会返回输

出的字符数，执行失败则返回 FALSE 和一个错误信息。

【示例 9-29】以下代码演示使用 readfile()函数输出文件内容。

```
<?php
    $texts=readfile('file.txt');              //输出文件内容并存储函数返回值
    echo "<br />该文件一个有{$texts}字符";     //输出函数返回的字符串个数
?>
```

代码运行结果如图 9.38 所示。

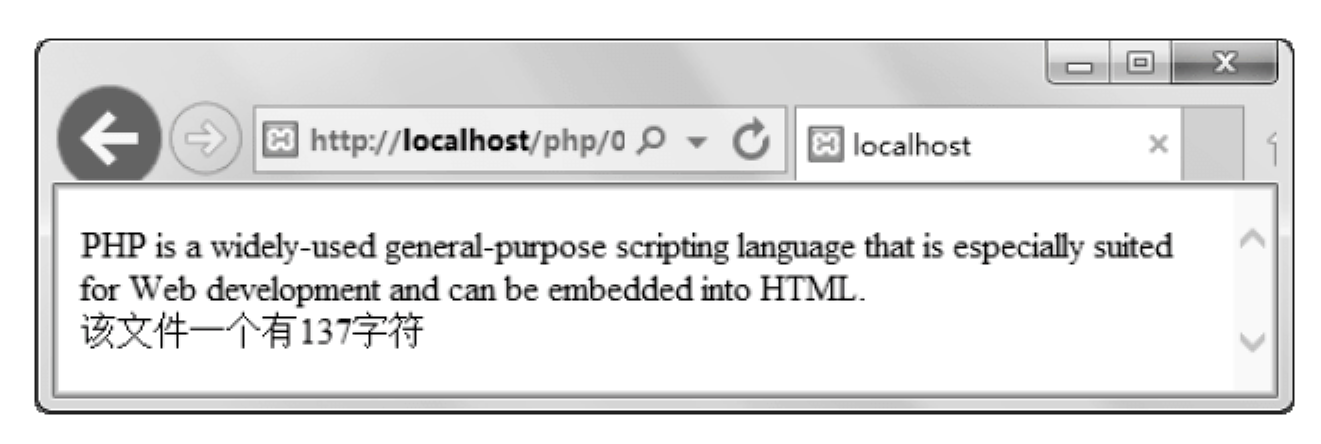


图 9.38　运行结果

该函数也可以用于读取一个 URL 并输出，这里我们就演示输出百度搜索的完整首页并输出该首页源文件的字符数。

【示例 9-30】以下代码演示使用 readfile()函数读取并输出百度搜索首页，以及该首页源文件的字符数。

```
<?php
    $texts=readfile('http://www.baidu.com');     //读取 URL 并统计字符数
    echo "<br />共输出了{$texts}个字符。";         //输出字符数
?>
```

代码运行结果如图 9.39 所示。



图 9.39　运行结果

从运行结果可以看到，使用 readfile()函数输出的界面与官方界面是无异的。

9.2.4　输出 PHP 代码

有时我们想要将一个 PHP 代码字符串或者文件输出，使用我们上面介绍的文件都可以

做到，但是效果不如使用我们将要介绍的两个函数好。highlight_string()函数可以用来使用 PHP 内置的语法高亮器来对 PHP 代码进行着色，highlight_file()函数则可以将一个文件中的代码进行着色后输出。这两个函数的原型如下：

```
mixed highlight_string ( string $str [, bool $return = false ] )
mixed highlight_file ( string $filename [, bool $return = false ] )
```

参数 str 为需要进行着色的代码；参数 filename 为需要进行着色的代码文件；可选参数 return 可以用来控制结果是否返回，默认为直接输出。

注意：需要润色的字符串或者文件开头必须为 PHP 开始标签“<?php”。

【示例 9-31】以下代码演示使用 highlight_string()函数对 PHP 代码字符串进行着色。

```
<?php
    $str="<?php echo 'Hello PHP.'?>";        //定义需要着色的字符串
    highlight_string($str);                  //着色字符串并输出
?>
```

代码运行结果如图 9.40 所示。由于黑白印刷原因读者不能看到着色的结果，这就需要读者亲自来编码验证了。

【示例 9-32】以下代码演示使用 highlight_file()函数着色输出 PHP 代码文件。

```
<?php
    highlight_file('./9-24.php');        //读取 PHP 代码文件并输出
?>
```

代码运行结果如图 9.41 所示。



图 9.40　运行结果



图 9.41　运行结果

为了简答演示，我们读取了本章前面所写的源代码。当然这也需要读者实际操作来查看运行的效果。

9.3　简单操作文件

与我们在平时的使用中类似的，可以使用 PHP 实现简单的文件操作。这里我们主要介绍复制、删除文件和重命名文件或者目录。由于在 PHP 中目前没有专门提供用于建立文件的函数，而建立文件通常是使用 fopen 间接建立文件，该函数将在 9.4 节中介绍。

9.3.1　复制文件

复制文件实现起来比较简单，可以使用 copy()函数来完成，它的原型如下：

```
bool copy ( string $source , string $dest )
```

参数 source 即为源文件的路径；参数 dest 即为复制文件的目标路径。copy()函数会在复制成功后返回 TRUE，失败则返回 FALSE。下面就来演示将示例所在当前目录下的一个文件，复制到当前目录下的 testdir 文件中。

【示例 9-33】以下代码演示将一个示例程序代码复制到同目录下的 testdir 目录中。

```
<?php
    $res=copy('./9-1.php','./testdir/9-1.php');
                                            //复制文件并保存函数返回结果
    if($res)                                //通过判断输出提示信息
        echo '文件复制成功。';
    else
        echo '文件复制失败。';
?>
```

代码运行结果如图 9.42 所示，提示复制成功，我们可以通过查看指定的路径来确认，如图 9.43 所示。



图 9.42　运行结果



图 9.43　复制后的文件

copy()函数可以在复制文件的时候进行重命名。

【示例 9-34】以下代码演示使用 copy()函数在复制文件的时候进行重命名。

```
<?php
    $res=copy('./9-1.php','./testdir/phpfile.php');
                                            //将文件复制为一个新名称的文件
    if($res)                                //通过判断输出提示信息
        echo '文件复制并重命名成功。';
    else
        echo '文件复制并重命名失败。';
?>
```

代码运行结果如图 9.44 所示。我们同样通过查看指定的路径来查看重命名后的文件，如图 9.45 所示。

我们可以查看这两个文件内容，如图 9.46 所示。

从两个文件的内容比较可以看出，这两个文件的内容还是相同的。



图 9.44　运行结果



图 9.45　复制后的文件



图 9.46　复制新名称的文件与源文件内容比较

9.3.2　重命名文件或者目录

虽然通过使用 copy()函数可以间接实现对文件的重命名，但是 PHP 提供了更加方便的 rename()函数来重命名文件或者目录。rename()函数的原型如下：

```
bool rename ( string $oldname , string $newname [, resource $context ] )
```

参数 oldname 为文件旧名称；参数 newname 为文件的新名称；可选参数 context 用来规定文件流的参数。该函数有两种使用效果，一种是将该文件进行重命名；另一种为将文件重命名的同时移动到指定的位置，类似于剪切文件的效果。下面我们来演示将 file.txt 文件重命名为 myfile.txt。

【示例 9-35】以下代码演示使用 rename()函数重命名文件。

```
<?php
    $res=rename('file.txt','myfile.txt');   //调用函数对文件进行重命名
    if($res)                                 //根据函数返回的结果输出提示信息
        echo '文件重命名成功。';
    else
        echo '文件重命名失败。';
?>
```

代码运行结果如图 9.47 所示。我们可以通过查看当前目录来验证重命名的效果，如图 9.48 所示。

从图 9.48 中可以看到函数使用的效果。下面再来演示将 myfile.txt 文件“剪切”到 textdir 目录中。

【示例 9-36】以下代码演示使用 rename()函数将 myfile.txt 文件“剪切”到 textdir 目录中。

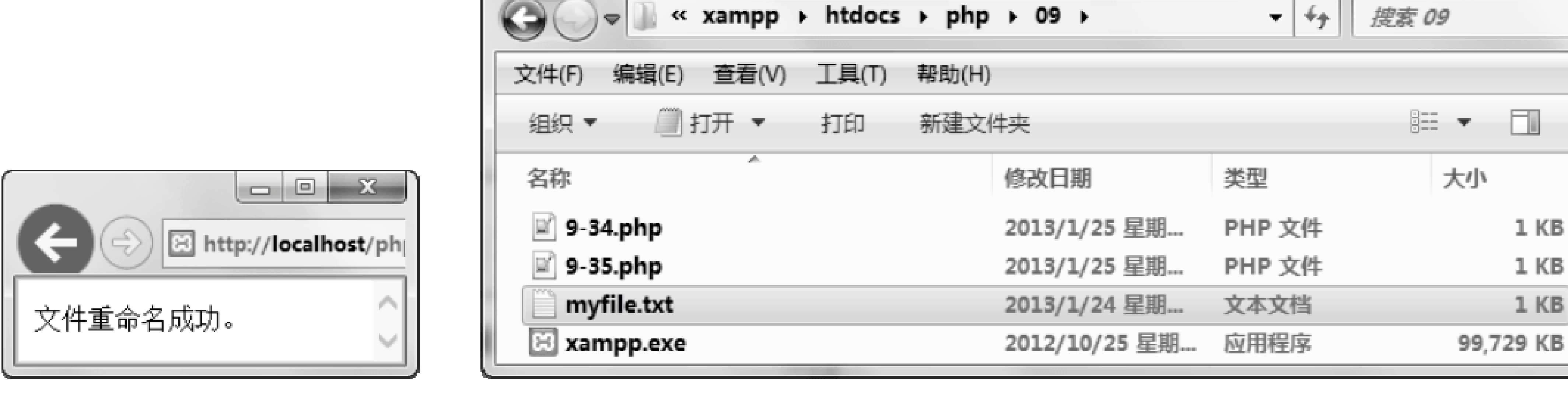


图 9.47　运行结果　　　　图 9.48　重命名的效果

```
<?php
    $res=rename('myfile.txt','./testdir/myfile.txt');  //对文件进行剪切
    if($res)                                         //通过运行结果输出提示信息
        echo '文件剪切成功。';
    else
        echo '文件剪切失败。';
?>
```

代码运行结果如图 9.49 所示。在执行程序后，我们将会看到如图 9.50 所示的效果。





图 9.49　运行结果　　　　图 9.50　剪切文件的效果

myfile.txt 文件被剪切到了 textdir 目录中，而原来路径下的 myfile.txt 文件则不再存在。

rename()函数不仅可以用来操作普通文件，还可以用来操作目录文件。下面我们就简单演示使用 rename()函数将 textdir 目录重命名为 mydir。

【示例 9-37】以下代码演示使用 rename()函数将 textdir 文件重命名为 mydir。

```
<?php
    $res=rename('testdir','mydir');      //对目录进行重命名
    if($res)                             //根据函数返回结果输出提示信息
        echo '目录重命名成功。';
    else
        echo '目录重命名失败。';
?>
```

代码运行结果如图 9.51 所示。我们可以通过查看该目录所在的路径验证执行效果，如图 9.52 所示。

从图 9.52 中可以看到，目录被正确重命名。关于使用 rename()函数对目录进行的其他操作文件的操作类似，这里不再详细介绍。



图 9.51　运行结果



图 9.52　目录重命名的效果

9.3.3　删除文件

在前面学习关于目录操作的过程中，我们已经了解了如何使用 rmdir()函数删除一个目录，但是非空目录需要首先将目录中的文件进行删除。现在我们就来介绍删除文件的函数 unlink()，它的原型如下：

```
bool unlink ( string $filename )
```

参数 filename 即为需删除的文件名称。该函数在执行成功后返回 TRUE，失败则返回 FALSE。我们首先来演示删除 mydir()目录中的 9-1.php 文件。

【示例 9-38】以下代码演示使用 unlink()函数删除 mydir 目录下的 9-1.php 文件。

```
<?php
    $res=unlink('./mydir/9-1.php');     //使用函数删除文件
    if($res)                            //根据函数的返回结果输出提示信息
        echo '文件删除成功。';
    else
        echo '文件删除失败。';
?>
```

代码运行结果如图 9.53 所示。我们可以查看 mydir 目录以确认函数的执行效果，如图 9.54 所示。



图 9.53　运行结果



图 9.54　代码执行效果

从图 9.54 中可以看到，9-1.php 文件已经不复存在。

下面我们就来演示一个前面遗留的问题——删除一个非空目录。首先来看要删除的目录结构，如图 9.55 所示。其中，mydir 目录下有 dir 目录和两个普通文件；dir1、dir2 和 dir3 目录均为空目录。

【示例 9-39】以下代码演示使用 unlink 联合 rmdir()函数删除一个非空目录。

```
<?php
    function deldir($dirname){  //定义函数名称
        $arr=scandir($dirname); //获取目录下文件信息
        foreach($arr as $v){    //循环遍历数组
            //判断文件名是否为目录并且不为特殊的“.”和“..”目录
            if(is_dir("{$dirname}/{$v}")&&$v!='.'&&$v!='..')
                deldir("{$dirname}/{$v}");
                                    //将目录作为递归调用 deldir 函数的参数
            elseif(is_file("{$dirname}/{$v}")){
                                    //如果不为目录则判断文件名是否为普通文件
                unlink("{$dirname}/{$v}");                  //删除文件
                echo "删除文件{$dirname}/{$v}<br />";      //输出提示信息
            }
        }
        rmdir($dirname);                                    //删除空目录
        echo "删除目录{$dirname}<br />";                   //输出提示信息
    }
    deldir('./mydir');                                      //调用函数
?>
```

代码运行结果如图 9.56 所示。



图 9.55　删除目录的结构



图 9.56　运行结果

从运行结果可以看出，程序执行后共删除了 4 个目录和两个普通文件。该代码中使用到的知识比较多，有函数的递归调用及相对路径的组合，读者即使不能理解也无妨，在水平提高以后自然就很容易写出这段代码了。

9.4　利用文件句柄操作文件

前面我们介绍了一些简单读取和输出文件，同目录文件操作类似，普通文件也可以通过文件句柄来灵活操作文件。

9.4.1　打开和关闭文件句柄

利用文件句柄来操作目录也需要首先打开一个文件句柄，然后进行对文件的操作，操作完成后再关闭文件句柄。打开文件句柄需要使用 fopen()函数，关闭文件句柄则需要 fclose()函数。这两个函数的原型如下：

```
resource fopen ( string $filename , string $mode [, bool $use_include_path
[, resource $context ]] )
bool fclose ( resource $handle )
```

fopen()函数中的 filename 即为需要操作的目标文件名；参数 mode 为操作文件的模式，该参数选项如表 9.2 所示。

表 9.2　fopen 参数选项

选项	说　明
'r'	只读方式打开，将文件指针指向文件头
'r+'	读写方式打开，将文件指针指向文件头
'w'	写入方式打开，将文件指针指向文件头并将文件大小截为 0。如果文件不存在则尝试创建它
'w+'	读写方式打开，将文件指针指向文件头并将文件大小截为 0。如果文件不存在则尝试创建它
'a'	写入方式打开，将文件指针指向文件末尾。如果文件不存在则尝试创建它
'a+'	读写方式打开，将文件指针指向文件末尾。如果文件不存在则尝试创建它
'x'	创建并以写入方式打开，将文件指针指向文件头。如果文件已存在，则 fopen()函数调用失败并返回 FALSE，并生成一条 E_WARNING 级别的错误信息。如果文件不存在则尝试创建它
'x+'	创建并以读写方式打开，将文件指针指向文件头。如果文件已存在，则 fopen()函数调用失败并返回 FALSE，并生成一条 E_WARNING 级别的错误信息。如果文件不存在则尝试创建它

可选参数 use_include_path 用于规定是否从 PHP 配置文件指定的路径查找文件；可选参数 context 用于规定数据流的参数。fclose()函数中的 handle 即为 fopen()函数打开的文件句柄。

【示例 9-40】以下代码演示使用 fopen()函数打开一个文件句柄，并使用 fclose()函数关闭打开的文件句柄。

```
<?php
    $fh=fopen('./file.txt','r');            //以只读方式打开 file.txt 文件
    //进行对文件的操作
    fclose($fh);                            //关闭打开的文件句柄
?>
```

以上代码只是进行了一系列的操作，因此在浏览器界面并不会有任何输出。从 fopen()函数的 mode 选项中也可以得知，使用不同的选项可以创建不存在的文件。下面我们就来演示使用 w 模式来创建一个不存在的文件。

【示例 9-41】以下代码演示使用 fopen()函数创建一个不存在的文件。

```
<?php
    function createfile($filename){            //创建一个函数
        if(!file_exists($filename)){           //判断文件名是否存在
            $fh=fopen($filename,'w');          //以 w 模式创建文件
            fclose($fh);                       //关闭打开的文件句柄
            echo "{$filename}创建成功！";        //提示创建成功
        }else
            echo "文件已经存在，不可创建！";       //文件存在则提示不可创建
    }
    createfile('createmyfile.txt');
?>
```

代码运行结果如图 9.57 所示。代码运行后提示文件创建成功，我们可以查看该文件，如图 9.58 所示。



图 9.57　运行结果



图 9.58　查看新创建的文件

当再次运行该示例代码的时候，浏览器会输出如图 9.59 所示的信息。



图 9.59　再次运行的结果

在打开一个文件时我们常常会判断文件是否已经存在，这是非常有必要的。如果 fopen()函数使用 w 或者 w+形式打开存在的文件时，会将这个文件长度截为 0，那么原来文件的信息就会丢失。

9.4.2　文件指针

同数组指针类似，文件中也有指针，在学习了数组指针之后，文件指针就非常容易理解了。首先我们来看文件指针操作的 3 个函数，它们的原型如下：

```
int ftell ( resource $handle )
int fseek ( resource $handle , int $offset [, int $whence ] )
bool rewind ( resource $handle )
```

ftell()函数用于返回文件指针的位置，参数 handle 为文件句柄。fseek()函数用于操作文件指针，参数 handle 为文件句柄；参数 offset 用来设置文件指针的偏移量，文件指针会随着打开的模式不同而指向文件的开头或者末尾；可选参数 whence 用于规定文件指针对于 offset 的操作，可能的选项如下。

- ❑ SEEK_SET：设定位置等于 offset（默认选项）；
- ❑ SEEK_CUR：设定位置为当前位置加上 offset；
- ❑ SEEK_END：设定位置为文件尾加上 offset（需要将 offset 设置为负值）。

rewind()函数用于将文件指针重新指向文件流的开头，参数 handle 为文件句柄。

下面就以一段简单的示例来演示使用以上 3 个函数操作文件指针的过程。

【示例 9-42】以下代码演示使用文件指针操作函数操作文件指针。

```
<?php
    $fh=fopen('file.txt','r');        //打开一个文件句柄
    echo "现在文件指针所在的位置为：".ftell($fh);
    fseek($fh,15);                    //移动文件指针
    echo "<br />移动文件指针后，文件指针所在的位置为：".ftell($fh);
    rewind($fh);                      //将指针重置
    echo "<br />将指针重置后，文件指针所在的位置为：".ftell($fh);
?>
```

代码运行结果如图 9.60 所示。

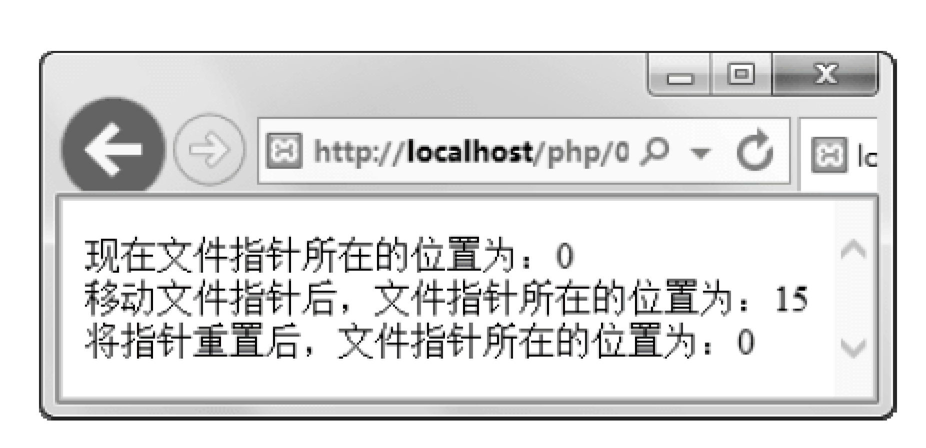


图 9.60　运行结果

文件指针操作函数使用非常简单，这里不再详细介绍。

9.4.3　读取文件操作

读取文件可以说是文件系统操作中主要的部分，但是使用起来并不会特别复杂，下面我们就来介绍读取文件操作的函数。

1. feof()函数

feof()函数用于测试文件指针是否到了文件结束的位置，该函数是在读取文件中常用到的函数，它的原型如下：

```
bool feof ( resource $handle )
```

参数 handle 即为文件句柄。该函数常用在循环读取文件的代码中。

2. 读取字符（串）函数

PHP 提供了多个读取字符或者字符串的函数。首先来看 fgetc()函数的原型：

```
string fgetc ( resource $handle )
```

fgetc()函数用于从文件中读取一个字符，遇到 EOF（end of file）则返回 FALSE。参数 handle 为文件句柄。

【示例 9-43】以下代码演示使用 fgetc()函数从文件中读取字符。

```
<?php
    $fh=fopen('./file.txt','r');          //打开文件句柄
    if(!feof($fh)){                       //判断是否为文件句柄
        echo '读取该文件的前 3 个字符为：';
        //三次调用 fgetc()函数读取文件前 3 个字符
        echo fgetc($fh);
        echo fgetc($fh);
        echo fgetc($fh);
    }
?>
```

代码运行结果如图 9.61 所示。



图 9.61　运行结果

从运行结果可以看到，程序在执行后输出了文件的前 3 个字符。我们可以使用循环来读取文件中指定长度的字符或者全部的字符。

【示例 9-44】以下代码演示使用 fgetc()函数结合循环

输出指定个数的字符串。

```
<?php
    //定义函数，filenam 为需要读取的文件名，$num 为读取的字符数
    function getchars($filename,$num=FALSE){
        $fh=fopen($filename,'r');   //打开文件句柄
        $i=0;                       //定义循环变量
        $chars='';                  //初始化变量
        while(!feof($fh)){          //判断文件指针是否指向文件结束
            if($num===$i)           //判断是否输出了特定的字符数
                break;              //跳出循环
            $chars.=fgetc($fh);     //将字符串循环保存到 chars 变量中
            $i++;                   //变量自增
        }
        return $chars;
        fclose($fh);
    }
    $filename='./file.txt';
    //调用函数输出指定个数的字符
    echo '输出文件的前 5 个字符：<br />'.getchars($filename,5);
    echo '<br /><hr />输出文件的前 15 个字符：<br />'
                                        .getchars($filename,15);
    echo '<br /><hr />输出文件的所有字符：<br />'.getchars($filename);
?>
```

代码运行结果如图 9.62 所示。



图 9.62　运行结果

代码中的自定义函数 getchars()会在使用默认参数时输出文件的所有字符。下面我们再来看 fgets()和 fgetss()函数，它们的原型如下：

```
string fgets ( resource $handle [, int $length ] )
string fgetss ( resource $handle [, int $length [, string $allowable_tags ]] )
```

fgets()和 fgetss()函数每次都会从文件中读取一行字符串并返回 length-1 长度的字符串，不同的是 fgetss()函数会将字符串中的 HTML 标签过滤掉。两个函数的 handle 参数为文件句柄；可选参数 length 用来规定要读取的字符串长度，默认为 1024 字节。但是并不是每次都会返回 length-1 长度的字符串，在碰到换行符和 EOF 时读取会提前结束。fgetss()函数的 allowable_tags 参数用来指定不进行过滤的标签。这两个函数的使用方法基本相同，我们就只介绍 fgets()函数。

【示例 9-45】以下代码演示使用 fgets()函数读取指定文件中的字符串。

```
<?php
```

```
        $fh=fopen('./file.txt','r');          //打开文件句柄
        $lines=0;                             //初始化记录行数的变量
        while(!feof($fh)){                    //判断是否到达文件末尾
            $lines++;                         //行数递增
            echo "文件中第{$lines}行的内容为: ".fgets($fh).'<br />';
                                              //输出字符串
        }
        fclose($fh);                          //关闭文件句柄
    ?>
```

代码运行结果如图 9.63 所示。由于我们的源文件中有多处换行，因此 fgets()函数会每次读取一行。下面我们再来演示规定每次只读取 15 个字符的情况。

【示例 9-46】以下代码演示使用 fgets()函数每次从文件中读取 15 个字符。

```
<?php
    $fh=fopen('./file.txt','r');          //打开文件句柄
    $lines=0;                             //初始化记录行数的变量
    while(!feof($fh)){                    //判断是否到达文件末尾
        $lines++;                         //行数递增
        echo "文件中第{$lines}行的内容为: ".fgets($fh,15).'<br />';
                                          //输出字符串
    }
    fclose($fh);                          //关闭文件句柄
?>
```

代码运行结果如图 9.64 所示。



图 9.63　运行结果



图 9.64　运行结果

从运行结果我们可以明显看到，第 8 行的内容只有 1 个字符，这就再次证明了不是每次都会返回 length-1 个字符。读者一定要注意这个特性。

fread()函数可以用来读取文件中指定长度的字符串，它的原型如下：

```
string fread (resource $handle , int $length )
```

参数 handle 即为文件句柄；参数 length 为需要读取的字符串的长度。该参数会在 3 种情况下结束：

- ❑ 读取了 length 个字节数的时候；
- ❑ 未达到 length 指定的字节数时提前遇到 EOF；
- ❑ 读取了 8192 个字节后。

该函数的使用和理解都非常简单，我们通过一个简单的示例来演示。

【示例 9-47】以下代码演示使用 fread()函数读取指定长度的字符串。

```
<?php
    $fh=fopen('./file.txt','r');          //打开文件句柄
    $chars=fread($fh,100);                //读取文件的前 100 个字符
    echo "读取文件中的前 100 字符为：<br />{$chars}";
    fclose($fh);                          //关闭文件句柄
?>
```

代码运行结果如图 9.65 所示。

虽然我们的源文件是作为多行保存的，但是该函数不会受到换行符的影响。

fpassthru()函数可以用来输出文件指针后所有剩余的数据，它的原型如下：

```
int fpassthru ( resource $handle )
```

参数 handle 为文件句柄。该函数在执行成功后会返回输出的字符数，执行失败则返回 FALSE。使用文件指针操作函数可以影响该函数的输出。

【示例 9-48】以下代码演示使用 fpassthru()函数输出文件指针后的剩余数据。

```
<?php
    $fh=fopen('./file.txt','r');          //打开文件句柄
    fseek($fh,100);                       //将文件指针向后移动 100
    echo '输出文件指针向后移动 100 后的内容：<br />';
    fpassthru($fh);                       //输出文件指针后剩余的内容
    fclose($fh);                          //关闭文件句柄
?>
```

代码运行结果如图 9.66 所示。

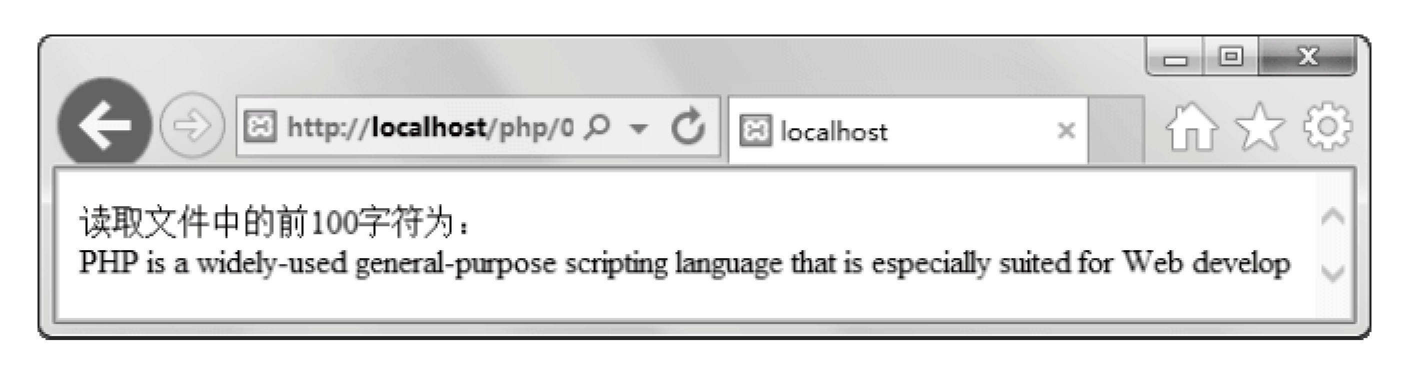


图 9.65　运行结果



图 9.66　运行结果

从运行结果可以看出该段示例输出了示例 9-47 输出内容的剩余部分。

9.4.4　写入文件操作

相对读取文件的函数来说，PHP 提供写入文件的函数是比较少的，这也就体现了写入文件并不是 PHP 常用的操作。常用的写入文件的函数是 fwrite()和 file_put_contents()函数，下面我们来分别介绍它们。

1. 使用 fwrite()函数

虽然 fwrite()函数本身的使用方法非常简单，但是随着打开文件的模式的不同，会有很不相同的表现形式。我们首先来看 fwrite()函数的原型：

```
int fwrite ( resource $handle , string $string [, int $length ] )
```

参数 handle 为打开的文件句柄；参数 string 为需要写入文件的内容；可选参数 length

用来规定写入的字符串长度。该函数在写完 length 长度或者字符串提前写完的时候会停止写入，成功后返回写入字符串的长度，失败则返回 FALSE。我们首先来演示新建一个文件并向其中写入内容。

【示例 9-49】以下代码演示使用 fopen()函数以“w+”模式打开文件，并使用 fwrite()函数向其中写入内容。

```
<?php
    $filename='writefile.txt';                          //定义目标文件名
    $str='PHP is a programming language.';              //定义写入内容
    if(!file_exists($filename))                         //判断文件是否存在
        $fh=fopen($filename,'w+');                      //以 w+模式打开文件
    $length=fwrite($fh,$str);                           //将内容写入文件
    fclose($fh);                                        //关闭打开的文件句柄
    echo "成功向{$filename}中写入了{$length}个字符。"; //输出写入后的信息
?>
```

代码运行结果如图 9.67 所示。从运行结果可以看出，程序运行后向文件中写入了数据。由于我们并没有显式地指定文件的路径，因此该文件会与源代码在同一目录下，我们可以找到并打开它，如图 9.68 所示。



图 9.67　运行结果

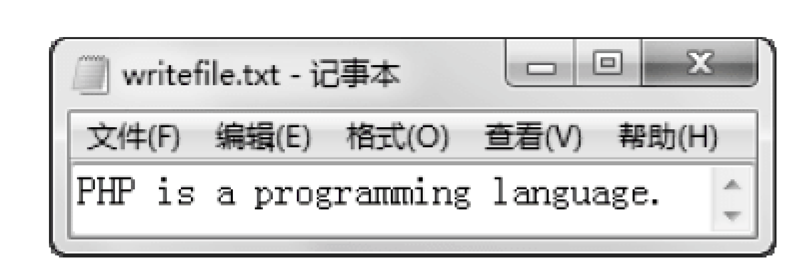


图 9.68　示例 9-49 生成的文件

下面我们来演示使用 fopen()函数以 a 模式打开示例 9-49 所创建的文件并向其写入内容。

【示例 9-50】以下代码演示使用 a 模式打开示例 9-49 创建的文件并向其中写入内容。

```
<?php
    $filename='writefile.txt';       //定义目标文件
    $str='PHP is easy to use.';      //定义写入内容
    if(file_exists($filename))       //判断文件是否存在
        $fh=fopen($filename,'a');    //以 a 模式打开文件
    $length=fwrite($fh,$str);        //向打开的文件句柄中写入内容
    fopen($fh);                      //关闭文件句柄
    echo "成功向{$filename}中写入了{$length}个字符。";   //输出写入后的信息
?>
```

代码运行结果如图 9.69 所示。

运行结果提示我们已经正确向文件中写入了内容。我们可以打开文件来查看写入的效果，如图 9.70 所示。



图 9.69　运行结果

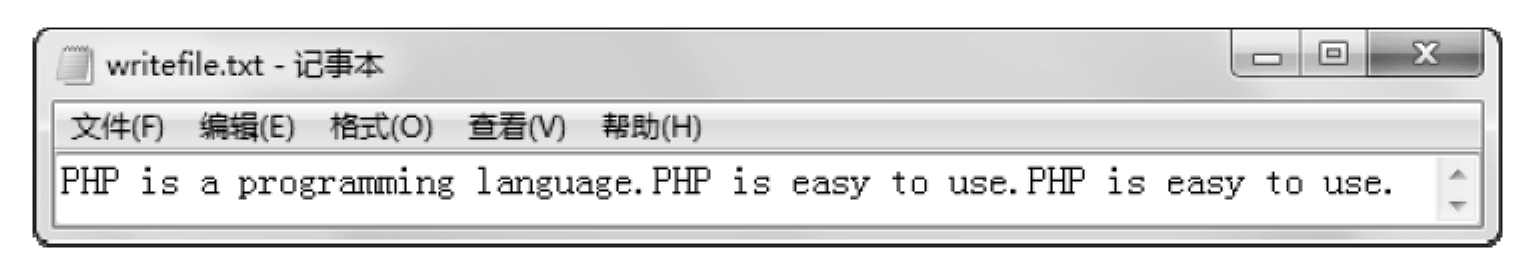


图 9.70　示例 9-50 写入后的文件内容

从运行结果中可以看出，示例 9-50 中的内容是以追加的形式向 writefile 文件中写入。

我们接着来看使用 fopen()函数以“r+”模式来打开文件并向其中写入内容。

【示例 9-51】以下代码演示使用 fopen()函数以“r+”模式打开文件并向其中写入内容。

```
<?php
    $filename='writefile.txt';                          //定义目标文件
    $str='Learning PHP is very interesting.';           //定义写入内容
    if(file_exists($filename))                          //判断文件是否存在
        $fh=fopen($filename,'r+');                      //以 a 模式打开文件
    $length=fwrite($fh,$str);                       //向打开的文件句柄中写入内容
    fclose($fh);                                    //关闭文件句柄
    echo "成功向{$filename}中写入了{$length}个字符。";   //输出写入后的信息
?>
```

代码运行结果如图 9.71 所示。



图 9.71　运行结果

运行结果提示我们向文件中写入内容成功。为了更方便读者进行比较，我们贴出示例 9-51 写入文件前后的 writefile 文件内容，如图 9.72 所示。

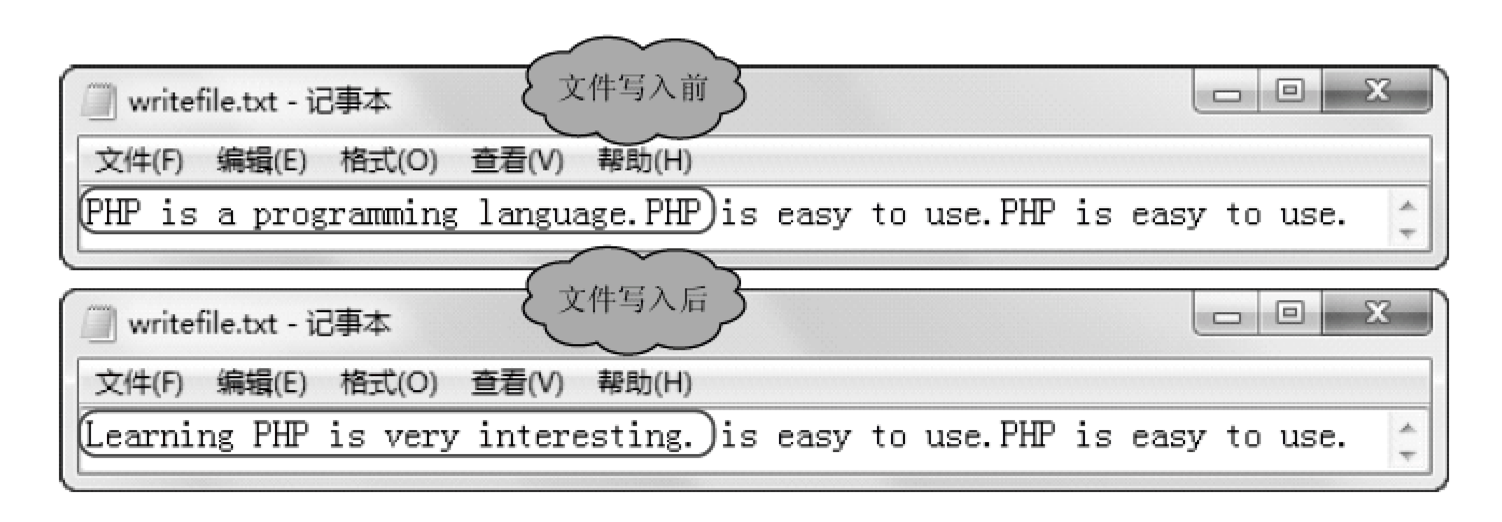


图 9.72　文件写入前后比较

从图 9.72 中可以看出，以“r+”模式打开的文件的文件指针是在文件开头的，因此写入的内容会覆盖文件中存在的内容。这种形式就类似于改写。以 w 模式打开文件都会将文件中原有的内容删除后进行写入操作，这里就不做详细介绍。

2. 文件锁定

文件锁定的概念也是比较容易理解的，因为一个文件有时候可能不只是由一个人进行写入操作。因此，当同时对一个文件进行写入操作的时候很可能会造成写入数据丢失。因此 PHP 提供了 flock()函数，在对文件进行写入操作的时候对文件进行锁定。flock()函数的原型如下：

```
bool flock (resource $handle , int $operation [, int &$wouldblock ] )
```

参数 handle 为打开的文件句柄；参数 operation 用于指定锁定的类型，可以为如下所示的值之一：

- ❑ LOCK_SH：共享锁定（读取的程序）；
- ❑ LOCK_EX：独占锁定（写入的程序）；

- LOCK_UN：释放锁定；
- LOCK_NB：不希望 flock()在锁定时堵塞时应该设置的选项。

可选参数 wouldblock 用来指定文件在锁定时是否阻塞其他对文件的操作。该函数会在成功锁定文件后返回 TRUE，如失败则返回 FALSE。flock()函数在 Windows 操作系统下会强制执行。锁定操作也可以由 fclose()函数释放。下面我们就来演示锁定一个文件。

【示例 9-52】以下代码演示是 flock 函数锁定一个文件。

```
<?php
    $filename='writefile.txt';      //锁定的目标文件
    if(file_exists($filename)){     //判断文件是否存在
        $fh=fopen($filename,'a');   //打开文件句柄
        if(flock($fh,LOCK_EX))      //判断是否锁定文件并输出提示
            echo "{$filename}文件已被成功锁定。";
        else
            echo "无法锁定{$filename}文件。";
        fclose($fh);                //关闭文件句柄并释放锁定
    }
?>
```

代码运行结果如图 9.73 所示。



图 9.73　运行结果

从运行结果可以看出，该文件已经被成功锁定，可以对其进行安全的写入操作了。

9.5　文件上传

文件上传在网络中是非常常用的操作。例如我们在网站注册账号的时候，通常会要求我们上传一个头像。本节我们就来简单介绍一下使用 PHP 实现文件上传。

9.5.1　配置环境

在进行文件的上传前，可能需要对 php.ini 文件进行一些配置，需要配置的项目及说明如下：

```
458 max_execution_time = 30            //每个脚本可以执行的最长时间
479 memory_limit = 128M                //每个脚本可以消耗的容量
913 file_uploads = On                  //是否允许上传文件
918 upload_tmp_dir = "D:\xampp\tmp"    //上传文件的临时目录
922 upload_max_filesize = 2M           //上传文件的最大容量
925 max_file_uploads = 20              //单个请求可以上传的最多文件数
```

在配置好允许环境后，我们需要认识一个预定义变量“$_FILES”。该变量是一个二维数组，它保存着上传文件的相关信息，该数组的详细信息如图表 9.3 所示。

表 9.3　预定变量$_FILES 数组信息

数组元素	保存的信息
$_FILES[filename][name]	保存上传文件的文件名
$_FILES[filename][size]	保存上传文件的大小
$_FILES[filename][tmp_name]	保存上传文件的临时名称
$_FILES[filename][type]	保存上传文件的类型
$_FILES[filename][error]	保存上传文件结果的代号，0 表示成功

该预定义变量会保存页面中所有上传的文件。

9.5.2　上传文件

上传文件需要使用到一些简单的HTML代码来建立一个文件上传页面。当然这些HTML代码可以直接写在 PHP 文件中或者由 PHP 函数进行输出。需要使用到的 HTML 代码如下：

```
<form enctype='multipart/form-data' method='post'>
    <input type='file' name='usrfile' />
    <input type='submit' value='确认上传'>
</form>
```

该代码建立一个如图 9.74 所示的页面。从图 9.74 中可以看出，这种形式的文件上传在很多网页中都见过。下面我们就来演示上传一个文件并遍历预定义变量$_FILES。

【示例 9-53】以下代码演示上传一个文件并遍历预定义变量$_FILES 获取文件信息。

```
<form enctype='multipart/form-data' method='post'>
    <input type='file' name='usrfile' />
    <input type='submit' value='确认上传'>
</form>
<?php
    echo '<pre>';
    if(!empty($_FILES))          //判断变量是否为空
        print_r($_FILES);        //遍历数组
    echo '</pre>'
?>
```

我们选择文件并确认上传后，代码的运行结果如图 9.75 所示。

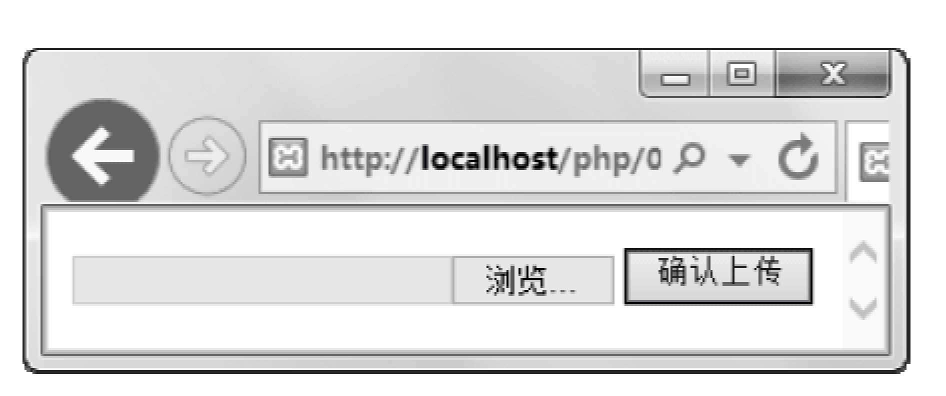


图 9.74　HTML 代码建立的页面



图 9.75　运行结果

从运行结果可以看出，此示例中的预定义变量$_FILES 中包含一个名为 usrfile 的元素，这个元素仍然为数组，其中包含了 usrfile 文件的信息。需要注意的是，虽然程序在执行过程中会生成临时上传的文件，但是会在程序执行结束后自动清除。因此我们不会看到临时文件。这时就需要将上传的临时文件存储起来。PHP 提供了 move_uploaded_file()函数来移动上传的文件，该函数的原型如下：

```
bool move_uploaded_file ( string $filename , string $destination )
```

参数 filename 为需要移动的文件名即临时文件名；参数 destination 为临时文件需要移动到的位置。为了方便演示，这里我们将临时文件保存到程序所在目录的 uploaded_file 文件夹中。

【示例 9-54】以下代码演示使用 move_uploaded_file()函数将上传的文件移动到指定的位置。

```
<form enctype='multipart/form-data' method='post'>
    <input type='file' name='usrfile' />
    <input type='submit' value='确认上传'>
</form>
<?php
    if(!empty($_FILES)){                                    //判断变量是否为空
        $filename=$_FILES['usrfile']['tmp_name'];  //定义要转移的文件名
        $filedir="./uploaded_file/{$_FILES['usrfile']['name']}";
                                                            //定义目标路径
        if(move_uploaded_file($filename,$filedir)) //判断是否转移成功
            echo "{$_FILES['usrfile']['name']}已经保存。";
                                                            //成功则输出提示
    }
?>
```

我们选择文件并确认上传后，代码的运行结果如图 9.76 所示。

运行结果提示 mypng.png 文件已经保存，可以通过我们保存文件的位置来查看，如图 9.77 所示。





图 9.76　运行结果



图 9.77　保存的上传文件

我们已经看到上传的文件已经在指定的文件夹中了。在有的情况下我们需要上传多个文件，下面我们就来演示上传 3 个文件的情况。我们首先需要改写一下原来的 HTML 代码，如下所示。

```
<form enctype='multipart/form-data' method='post'>
    <input type='file' name='usrfile1' /><br />
    <input type='file' name='usrfile2' /><br />
```

```
    <input type='file' name='usrfile3' /><br />
    <input type='submit' value='确认上传'>
</form>
```

我们同样通过输出预定义变量“$_FILES”数组的内容，来查看上传多个文件时候的情况。

【示例 9-55】以下代码演示上传 3 个文件并输出预定义变量“$_FILES”的信息。

```
<form enctype='multipart/form-data' method='post'>
    <input type='file' name='usrfile1' /><br />
    <input type='file' name='usrfile2' /><br />
    <input type='file' name='usrfile3' /><br />
    <input type='submit' value='确认上传'>
</form>
<?php
    echo '<pre>';
    if(!empty($_FILES))          //判断变量是否为空
        print_r($_FILES);        //输出变量信息
    echo '</pre>';
?>
```

我们选择文件并确认上传后，代码的运行结果如图 9.78 所示。

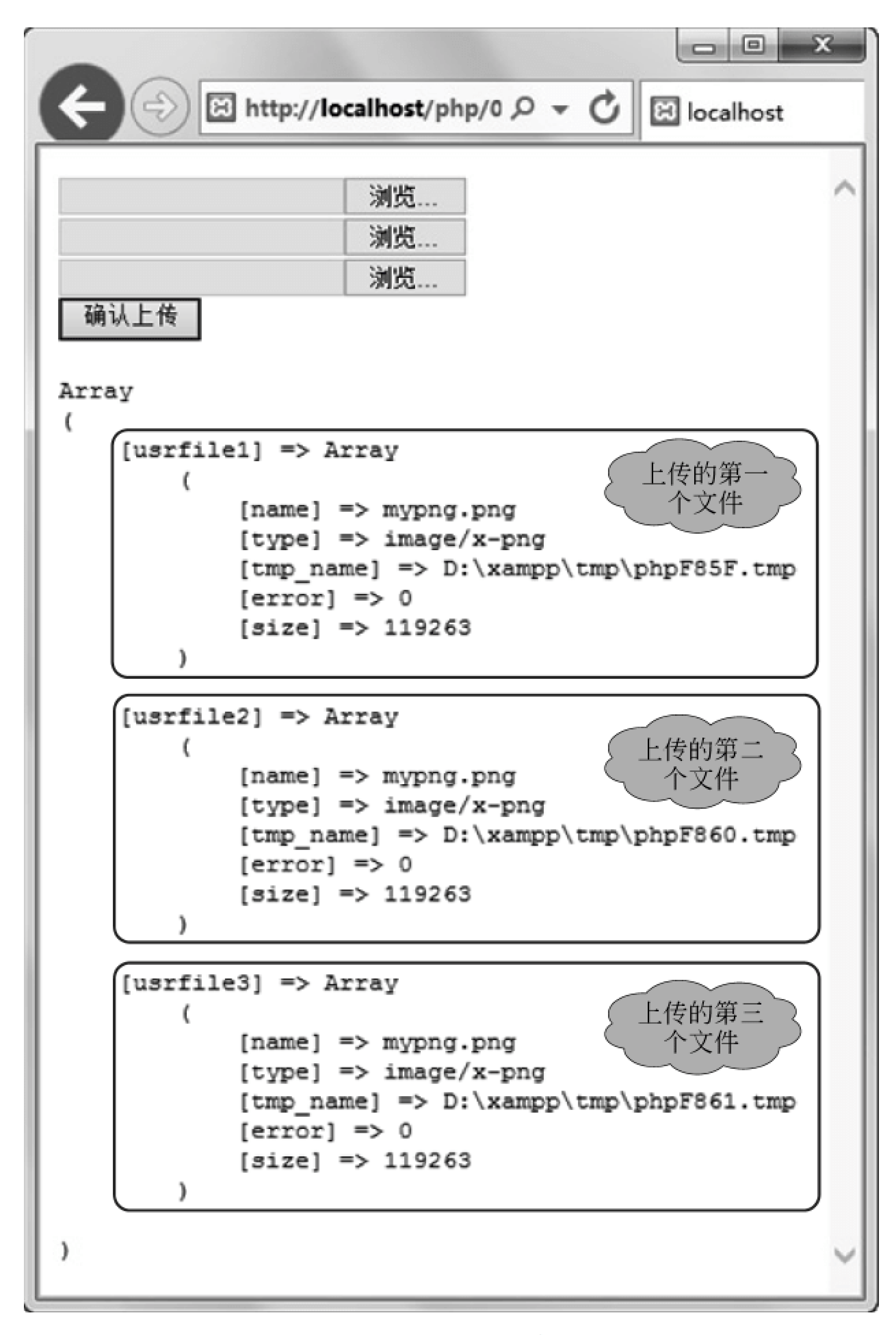


图 9.78　运行结果

从运行结果可以看到，预定义变量“$_FILES”获取到了上传文件的信息。在实际的

情况下，HTML 部分的代码会使用一个 HTML 的特性，示例中的 HTML 代码会写成如下的形式：

```
<form enctype='multipart/form-data' method='post'>
    <input type='file' name='usrfile[ ]' /><br />
    <input type='file' name='usrfile[ ]' /><br />
    <input type='file' name='usrfile[ ]' /><br />
    <input type='submit' value='确认上传'>
</form>
```

这种形式就类似于数组，但使我们不必定义太多的名称，但是这种形式得到的结果与示例 9-55 的运行结果是不相似的。

【示例 9-56】以下代码演示使用改进后的 HTML 页面上传多个文件，并使用预定义全局变量获取上传文件的信息。

```
<form enctype='multipart/form-data' method='post'>
    <input type='file' name='usrfile[]' /><br />
    <input type='file' name='usrfile[]' /><br />
    <input type='file' name='usrfile[]' /><br />
    <input type='submit' value='确认上传'>
</form>
<?php
    echo '<pre>';
    if(!empty($_FILES))          //判断变量是否为空
        print_r($_FILES);        //输出变量信息
    echo '</pre>';
?>
```

选择文件并确认上传后，运行结果如图 9.79 所示。

从运行结果可以看到，预定义变量只获取到了一个文件名称，但是各个属性均有 3 个。这与示例 9-55 中获取到 3 个文件是完全不同的。在该示例中，我们可以看到只有一个文件名，而且各个文件的属性都存放在数组中，因此我们可以方便地使用循环来保存这些上传的文件。

【示例 9-57】以下代码演示将多个上传的文件保存在指定位置。

```
<form enctype='multipart/form-data' method='post'>
    <input type='file' name='usrfile[]' /><br />
    <input type='file' name='usrfile[]' /><br />
    <input type='file' name='usrfile[]' /><br />
    <input type='submit' value='确认上传'>
</form>
<?php
    if(!empty($_FILES)){              //判断变量是否为空
        for($i=0;$i<3;$i++){          //循环 3 次保存三个文件
            $filename=$_FILES['usrfile']['tmp_name'][$i];
                                      //定义要转移的文件名
            $filedir="./uploaded_file/{$_FILES['usrfile']['name'][$i]}";
                                      //定义目标路径
            if(move_uploaded_file($filename,$filedir))//判断是否转移成功
                echo "{$_FILES['usrfile']['name'][$i]}已经保存。<br />";
                                      //成功则输出提示
        }
    }
?>
```

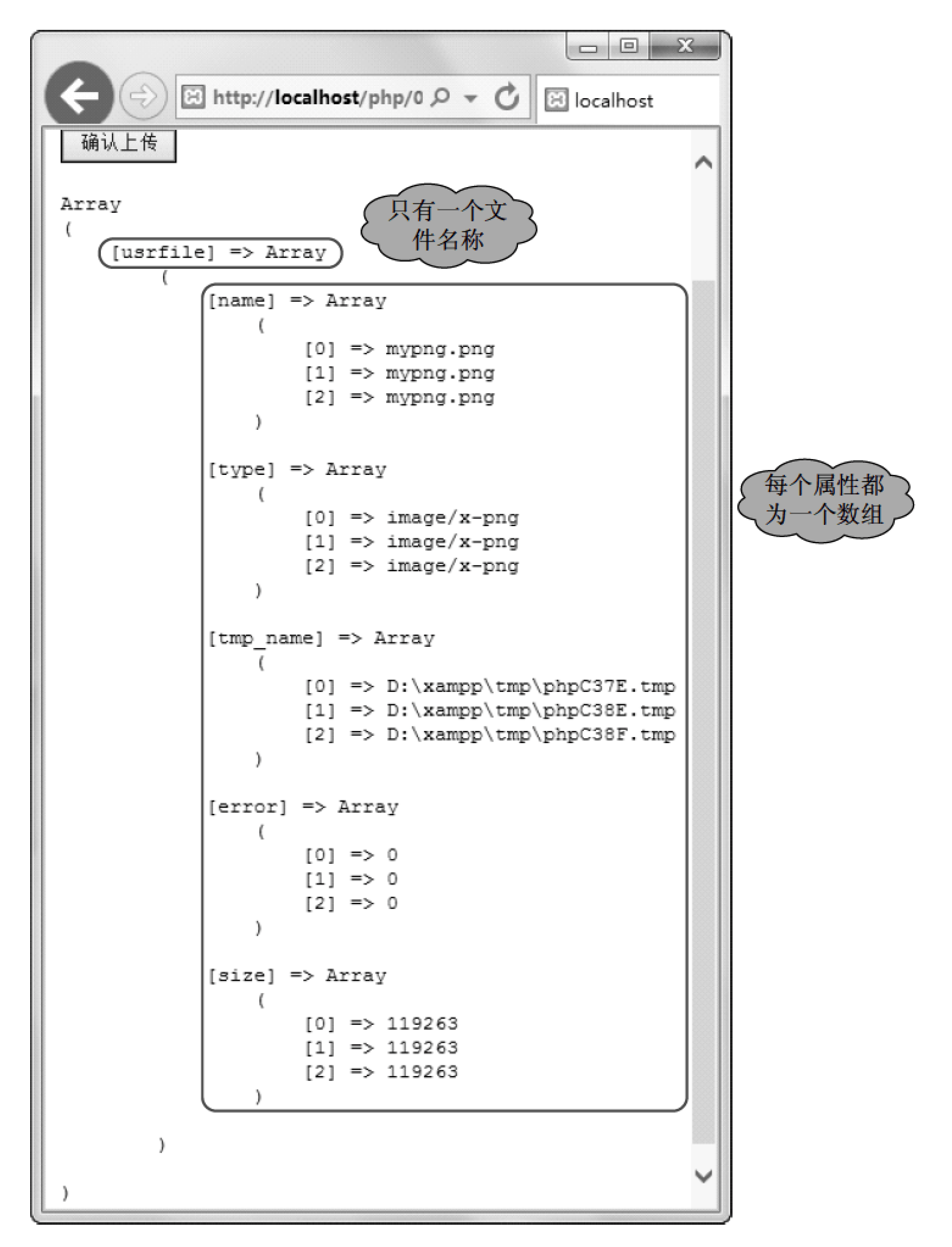


图 9.79 运行结果

代码运行结果如图 9.80 所示。

运行结果已经提示上传的文件都保存成功，我们可以查看指定路径，如图 9.81 所示。




图 9.80 运行结果



图 9.81 保存的文件

可以确认我们上传的文件都被正确保存了。文件上传的基本原理我们就全部介绍完毕了，当然以上示例可以使用不同形式的循环来灵活变化，例如使用 while 循环来保存上传的文件等。

9.6　小　　结

本章主要介绍了 PHP 文件系统的操作。虽然从整体篇幅上来看，讲解的内容不少，但是都比较容易理解，而且我们尽量地保证讲解得通俗易懂。在实际应用中要尽量考虑周全，例如在尝试读取文件或者写入文件的时候首先判断文件是否存在等。本章的这些知识通过实践后均可很好地掌握。

9.7　本 章 习 题

1. 在 D 盘新建一个文件夹，并在该文件夹新建名为 mytxt.txt 和 myphp.php 的文件，然后使用 PHP 输出该目录下的文件和目录。

2. 向习题 1 中创建的 mytxt.txt 文件写入“这是一个普通文本文件”；myphp.php 文件中写入如下的代码：

```
<?php
    echo "这是一个 PHP 文件";
?>
```

3. 获取习题 2 中修改的两个文件的修改时间。

4. 利用文件指针读取习题 2 中修改的两个文件中的内容。

第 10 章　图 像 处 理

PHP 不仅可以用来处理文字，而且支持很多种图像格式的处理。我们不仅可以处理已经存在的图片，而且可以自己画出一幅图。在网站中，有大量的图片需要由网站动态生成，如验证码图片，水印图片。PHP 提供了 GD 库，可以很轻松地实现这些功能。本章将详细讲解图像绘制及简易图片处理的知识。

10.1　处理图像前的准备

在进行本节知识的讲解前，读者需要做一些必要的环境配置及通用的图像知识，例如图像中的坐标系统等。读者只有配置好环境及了解必要的图像知识后，才可以轻松地理解和使用图像绘制相关的函数。

10.1.1　加载 GD 库

GD 库是 PHP 的图形扩展库。所谓扩展就是非必须的，PHP 不使用它也不会出现问题，但是使用它可以使一些工作更加容易，例如我们的图像处理。GD 库已经成为 PHP 默认安装的库，省去了我们进行安装。但是默认情况下 GD 库是没有加载的，需要通过配置 PHP 配置文件来加载。需要修改 php.ini 文件的如下设置：

```
extension=php_gd2.dll
```

我们需要做的就是将其前面的注释符号（;）去掉即可，然后重启服务器即可正确加载。当然由于我们使用的是集成开发环境，该扩展库已经默认被加载了。我们可以通过 phpinfo 或者 gd_info()函数来获取当前使用的 GD 库信息。

【示例 10-1】以下代码演示使用 gd_info()函数获取当前 GD 库的信息。

```
<?php
    echo '<pre>';            //HTML 标签，用来格式化输出以便读者查看
    print_r(gd_info());      //输出 gd_info()函数返回的数组信息
?>
```

代码运行结果如图 10.1 所示。

从运行结果可以看出，该函数返回的数组显示了已经加载的 GD 版本及对图像格式的支持情况。如果 GD 库没有被正确加载是不会出现这些信息的，读者可以以此来判断 GD 库的加载情况。

10.1.2　指定正确的 MIME 类型

MIME 是多用途 Internet 邮件扩展的缩写。现在它可以说是 Internet 内容类型描述的事

```
Array
(
    [GD Version] => bundled (2.0.34 compatible)
    [FreeType Support] => 1
    [FreeType Linkage] => with freetype
    [T1Lib Support] =>
    [GIF Read Support] => 1
    [GIF Create Support] => 1
    [JPEG Support] => 1
    [PNG Support] => 1
    [WBMP Support] => 1
    [XPM Support] =>
    [XBM Support] => 1
    [JIS-mapped Japanese Font Support] =>
)
```

图 10.1　GD 库信息

实标准。PHP 默认的 MIME 类型是 text/html，这在 PHP 配置文件中可以设置。Web 服务器在发送被请求内容到浏览器之前会首先发送一个文件头。我们可以通过设置这个文件头来使浏览器正确识别图像。这里需要使用到的函数是 header()函数，它的原型如下：

```
void header ( string $string [, bool $replace = true [, int
$http_response_code ]] )
```

参数 string 为要发送的报头字符串；可选参数 replace 用来规定是否替换原来的报头；可选参数 http_response_code 用来指定 HTTP 的响应代码。我们在输出图像的时候只需设置报头中 Content-Type 的内容。常用的图像 MIME 类型如表 10.1 所示。

表 10.1　图像 MIME 类型

图 像 类 型	MIME 表示
JPEG 文件可交换格式（.jpeg/.jpg）	image/jpeg
可移植网络图像（.png）	image/png
可交换图像格式（.gif）	image/gif
Windows 位图（.bmp）	image/bmp

例如，可以使用如下代码让浏览器以 png 格式的要求解析请求：

```
header('Content-Type:image/png')
```

需要注意的是，在设置的头文件输出前不可以有任何字符的输出，否则会导致图像无法显示。

10.1.3　通用图像知识

在图像处理的过程中，坐标系统占有非常重要的地位，如果不能熟练地掌握坐标系统，那么在使用图像处理函数的时候会非常吃力。

1. 图像坐标

通常情况下，图像的坐标系统与我们数学中使用的坐标系统是不同的。图像的坐标系统如图 10.2 所示。

图片尺寸通常使用像素来表示，单位为 px。假设我们上面定义的图片为 200×300 像素，那么它的宽度就是 200px，高度就是 300px，则它的一些常用的坐标如图 10.3 所示。

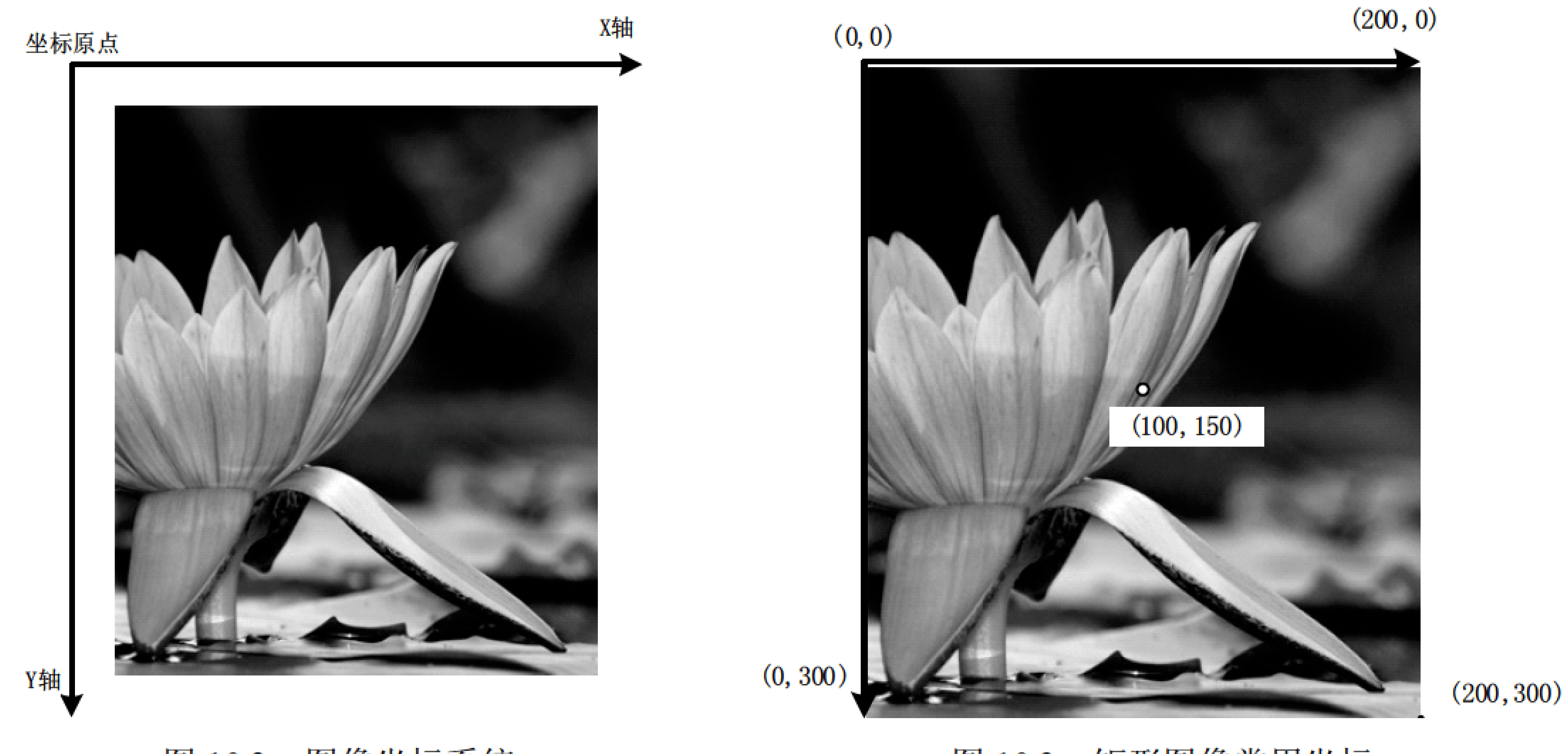


图 10.2　图像坐标系统　　　　图 10.3　矩形图像常用坐标

在进行简单会话的时候，坐标的掌握是非常重要的。只有非常熟练地掌握坐标系统，才能画出自己满意的效果。

2. 角度系统

角度在图形图像处理中常用在画弧线或者旋转图像之用。在编程领域通常使用的角度系统如图 10.4 所示。

也就是说，0° 在 3 点钟的位置，角度依次按照顺时针方向增加。

3. 绘制椭圆

在绘制椭圆的时候不存在半径参数。因此我们也需要了解一下画椭圆时需要使用到的参数。它的形式如图 10.5 所示。

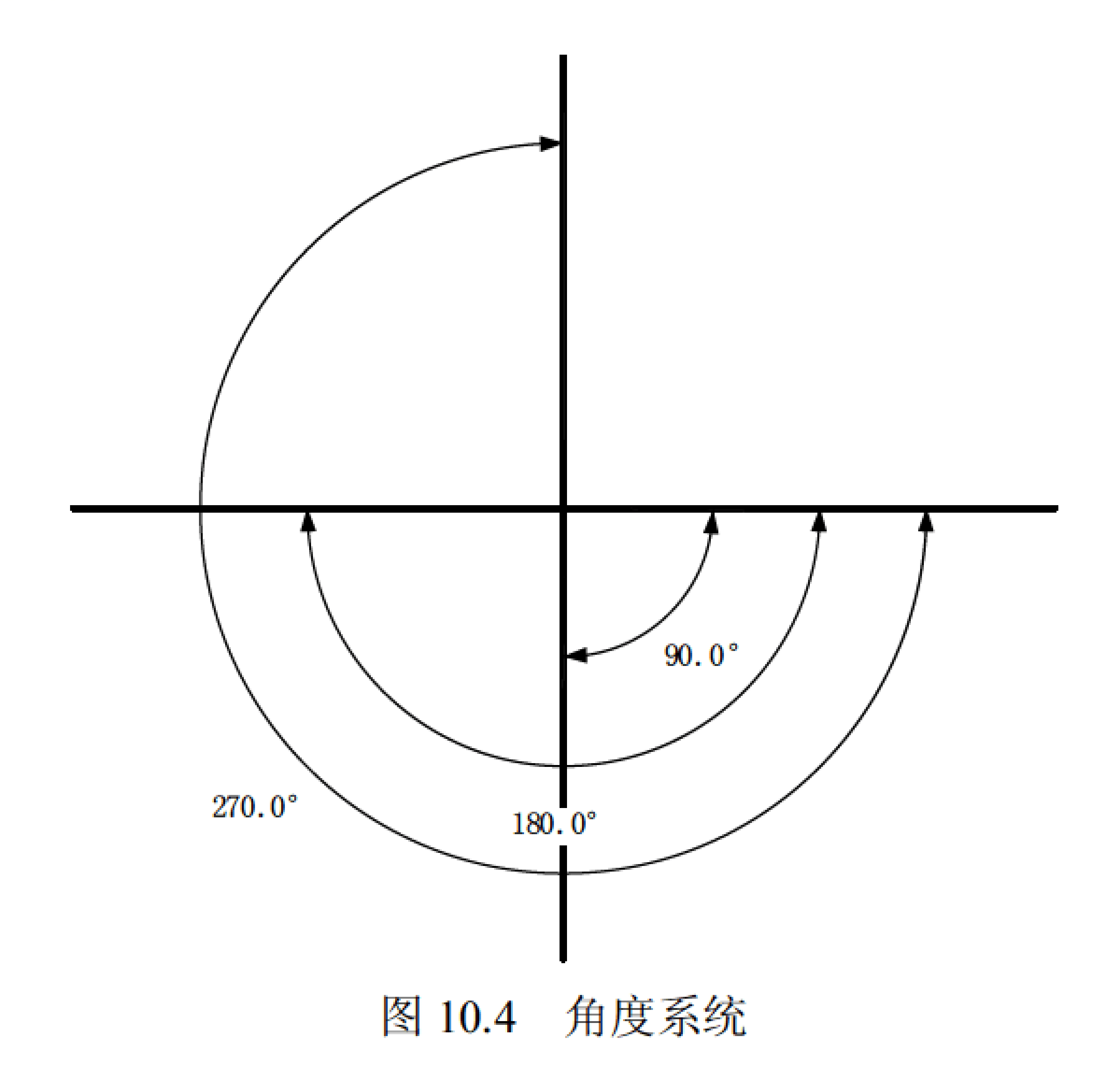


图 10.4　角度系统

宽度
高度
圆心
坐标

图 10.5　画需要的椭圆参数

一般在电脑中绘制椭圆时，认为椭圆是与相对应的矩形相切的，矩形的长即为椭圆长轴的长度，矩形的宽即为椭圆短轴的长度。所以通过矩形可以更好地理解椭圆的绘制，确定了矩形的位置及大小就可以确定椭圆的位置及大小。在不使用半径的情况下，在一个矩形的区域中想要定位一个椭圆是比较使初学者困惑的。例如，想要完成一个如图 10.6 所示的图像并不简单。

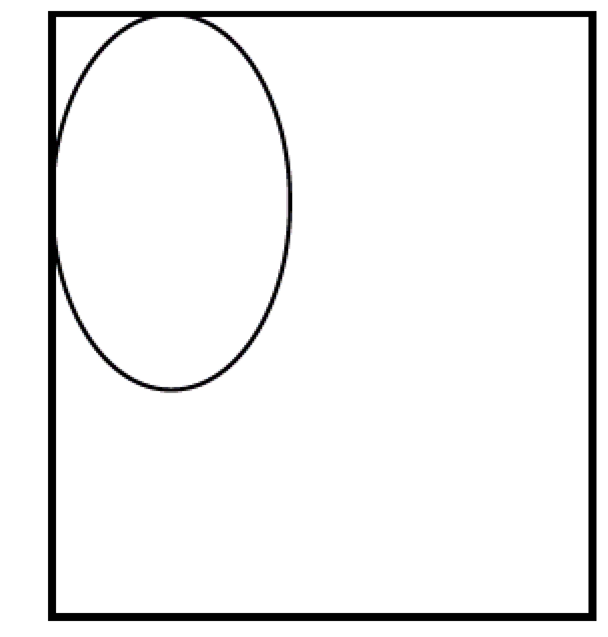


图 10.6　初学者不容易画出的图

想要画出图 10.6 中所示的图像的难点就在于椭圆圆心的确定，因此以上介绍的这些知识都是必须要去掌握的。

4. RGB 色彩模式

在图像处理的过程中，我们主要使用的色彩模式为 RGB 模式。RGB 即分别代表红（red）、绿（green）和蓝（blue）3 种颜色。这种模式使用 RGB 模型，为图像中每一个像素的 RGB 分量分配一个 0～255 范围内的强度值。也就是说，我们需要分别指定这 3 种颜色在一个像素点中的值，这些值可以为 0～255 中的一个或者其十六进制表示，如表 10.2 所示。

表 10.2　几种颜色以及对应 RGB 值

色彩	RGB 值
白色	（255,255,255）或者“#FFFFFF”
黑色	（0,0,0）或者“#000000”
红色	（255,0,0）或者“#FF0000”
绿色	（0,255,0）或者“#00FF00”
蓝色	（0,0,255）或者“#0000FF”

10.2　图 像 绘 制

经过 10.1 节的学习，基础的准备工作我们已经介绍完毕了。接下来就回到我们 PHP 知识的学习中。本节将会从创建画布开始介绍图像绘制的所有步骤。

10.2.1　输出图像的三个步骤

同现实中绘画类似，在绘画之前需要首先有一张“画布”然后我们在上面作画。在编程中，图像的输出通常需要 4 个步骤，分别为创建画布、绘制图像、输出图像和清除图像。下面我们就首先介绍绘制图像之外的 3 个步骤。

1. 创建空画布

PHP 中创建画空白布使用 imagecreate()和 imagecreatetruecolor()函数来创建，这两个函数的区别是 imagecreate()函数会创建一个基于调色板的图像，可以显示的色彩数通常为 256 色；而 imagecreatetruecolor 函数会创建一个真彩色的图像，通常可以显示 16777216 种颜色，官方推荐使用 imagecreatetruecolor()函数。但是 imagecreatetruecolor()函数不支持 GIF 格式的图像。这两个函数的原型如下：

```
resource imagecreate ( int $x_size , int $y_size )
resource imagecreatetruecolor ( int $x_size , int $y_size )
```

参数 x_size 即为需要创建的画布宽度；参数 y_size 为需要创建的画布高度，这两个函数会返回一个资源以供后续的操作使用。

注意：虽然 imagecreate()和 imagecreatetruecolor()函数都会创建一个空画布，但是默认情况下 imagecreate()函数会创建一个空白的画布，而 imagecreatetruecolor()函数则会创建出一个黑色的画布。

2. 创建基于文件或 URL 的画布

创建一个基于文件或 URL 的画布就类似于我们将一张图像作为背景然后在上面作画。由于 PHP 支持多种格式的图像类型，因此有多个函数用来创建不同文件格式的画布。常用的基于文件或 URL 创建画布的函数原型如下：

```
resource imagecreatefromgif ( string $filename )
resource imagecreatefromjpeg ( string $filename )
resource imagecreatefrompng ( string $filename )
```

参数 filename 为文件名或者 URL，这 3 个函数的返回值同样为资源类型。

3. 输出图像

同创建画布类似，由于 PHP 支持多种图像类型，因此也有多个函数用来输出对应格式的图像。常用的输出图像的函数原型如下：

```
bool imagegif ( resource $image [, string $filename ] )
bool imagejpeg ( resource $image [, string $filename [, int $quality ]] )
bool imagepng ( resource $image [, string $filename ] )
```

参数 image 为创建画布的函数返回的资源；可选参数 filename 用来规定是否以文件的形式保存而不直接输出。imagejpeg()函数的可选参数 quality 用来设置输出图像的质量，范围可为 0～100，默认大约为 75。

4. 清除图像资源

在图像输出后，就需要清除图像来释放资源，就类似于关闭文件句柄。清除图像使用的函数为 imagedestroy()，它的原型如下：

```
bool imagedestroy ( resource $image )
```

参数 image 即为创建画布函数返回的资源。下面我们就来演示从创建画布到清除图像的整个流程以及执行的效果。

【示例 10-2】以下代码演示使用 imagecreatetruecolor()函数创建一个空画布并输出。

```
<?php
    header('Content-Type:iamge/png');          //定义报头
    $img=imagecreatetruecolor(200,300);        //创建画布
    imagepng($img);                            //输出画布
    imagedestroy($img);                        //清除画布释放资源
?>
```

代码运行结果如图 10.7 所示。

从运行结果可以看出，使用 imagecreatetruecolor()函数会创建出一个黑色的画布。下面我们就来演示基于文件来创建一个画布。

【示例 10-3】以下代码演示使用 iamgecreatefromjpeg 函数创建一个基于文件的画布并输出。

```
<?php
    header('Content-Type:image/jpeg');           //定义报头
    $img=imagecreatefromjpeg('image.jpg');       //创建画布
    imagejpeg($img);                             //输出画布
    imagedestroy($img);                          //清除画布释放资源
?>
```

注意：在进行图像处理时，自定义的报头、创建画布的函数和输出图像的函数都要一致。例如创建的画布为 png 格式，那么发送的报头就必须为“Content-Type:image/png”，而且输出函数也必须使用 imagepng()函数。

代码运行结果如图 10.8 所示。

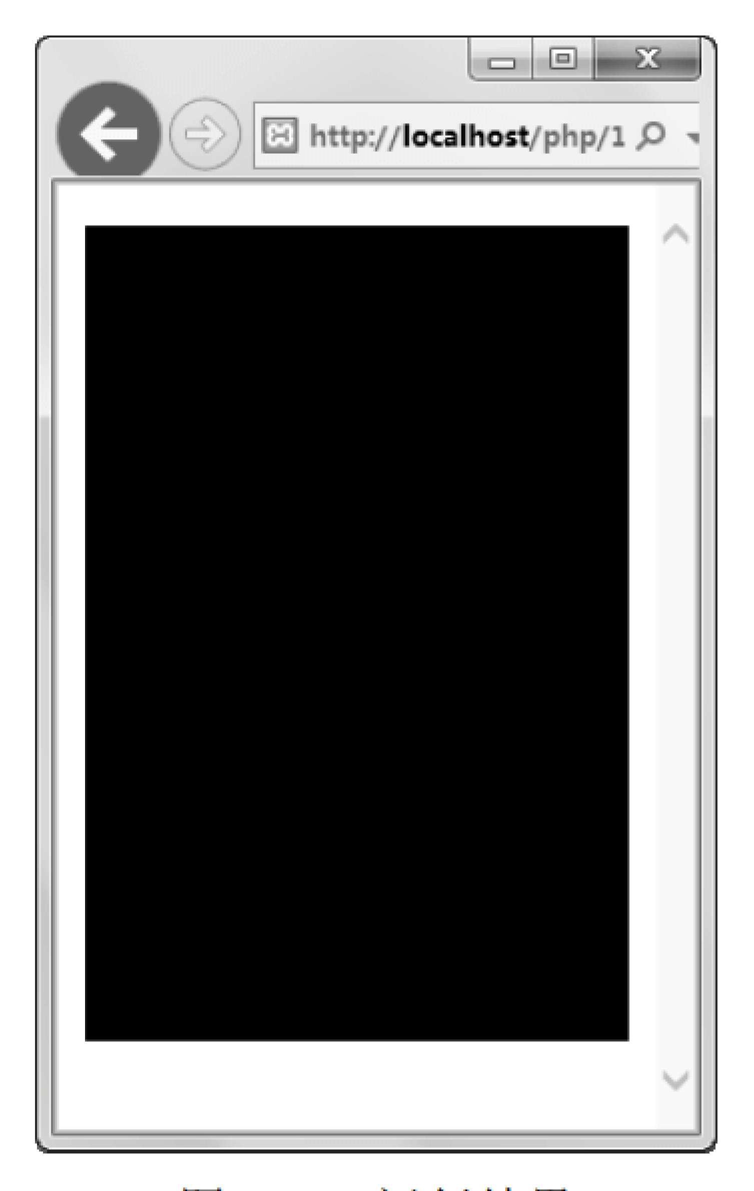


图 10.7　运行结果



图 10.8　运行结果

可以看到，本地的图像已经被正确输出了。下面再来演示创建一个基于 URL 的画布并以文件的形式保存。

【示例 10-4】以下代码演示使用 imagecreatefrompng()函数创建一个基于 URL 的画布并将其保存。

```
<?php
    $img=imagecreatefrompng('http://www.google.com.hk/images/srpr/
                        logo3w.png');            //创建画布
    $res=imagepng($img,'google_logo.png');       //输出画布
    imagedestroy($img);                          //清除画布释放资源
    if($res)                                     //保存成功则输出提示信息
        echo '图像已经被作为文件保存。';
?>
```

代码运行结果如图 10.9 所示。

运行结果已经提示图像已经被保存，我们可以通过查看目录来确认。打开保存的 google_logo.png 文件如图 10.10 所示。



图 10.9　运行结果



图 10.10　示例 10-4 程序运行后保存的文件

10.2.2　定义颜色

在创建了画布后，现在还不可以绘画，因为现在还没有创建颜色，这就类似于作画前调色的过程。定义颜色可以使用 imagecolorallocate()和 imagecolorallocatealpha()函数，它们的原型如下：

```
int imagecolorallocate ( resource $image , int $red , int $green , int $blue )
int imagecolorallocatealpha ( resource $image , int $red , int $green , int
$blue , int $alpha )
```

参数 image 为创建画布函数返回的资源；参数 reb、green、blue 分别用来定义 RGB 色的成分。imagecolorallocatealpha()函数中的参数 alpha 用来规定颜色的透明度，可用的值为 0～127，0 表示完全不透明，127 表示完全透明。

注意：imagecolorallocate()和 imagecolorallocatealpha()函数第一次定义的颜色会作为 imagecreate()函数创建的画布背景色。

【示例 10-5】 以下代码演示使用 imagecolorallocate()函数为 imagecreate()函数创建的画布设置背景色。

```
<?php
    header('Content-Type:image/png');         //定义报头
    $img=imagecreate(200,300);                //创建画布
    $black=imagecolorallocate($img,0,255,255);
                                              //定义青色并作为画布的背景色
    imagepng($img);                           //输出画布
    imagedestroy($img);                       //清除画布释放资源
?>
```

代码运行结果如图 10.11 所示。

从图中可以看到，基于调色板的画布被设置成为了黑色背景。下面我们再来演示使用 imagecolorallocatealpha()函数为颜色设置透明度。

【示例 10-6】以下代码演示使用 imagecolorallocatealpha()函数设置颜色透明度。

```
<?php
    header('Content-Type:image/png');       //定义报头
    $img=imagecreate(200,300);  //创建画布
    $black=imagecolorallocatealpha($img,0,0,0,100);
                                //定义黑色并设置透明度后作为画布的背景色
    imagepng($img);             //输出画布
    imagedestroy($img);         //清除画布释放资源
?>
```

代码运行结果如图 10.12 所示。

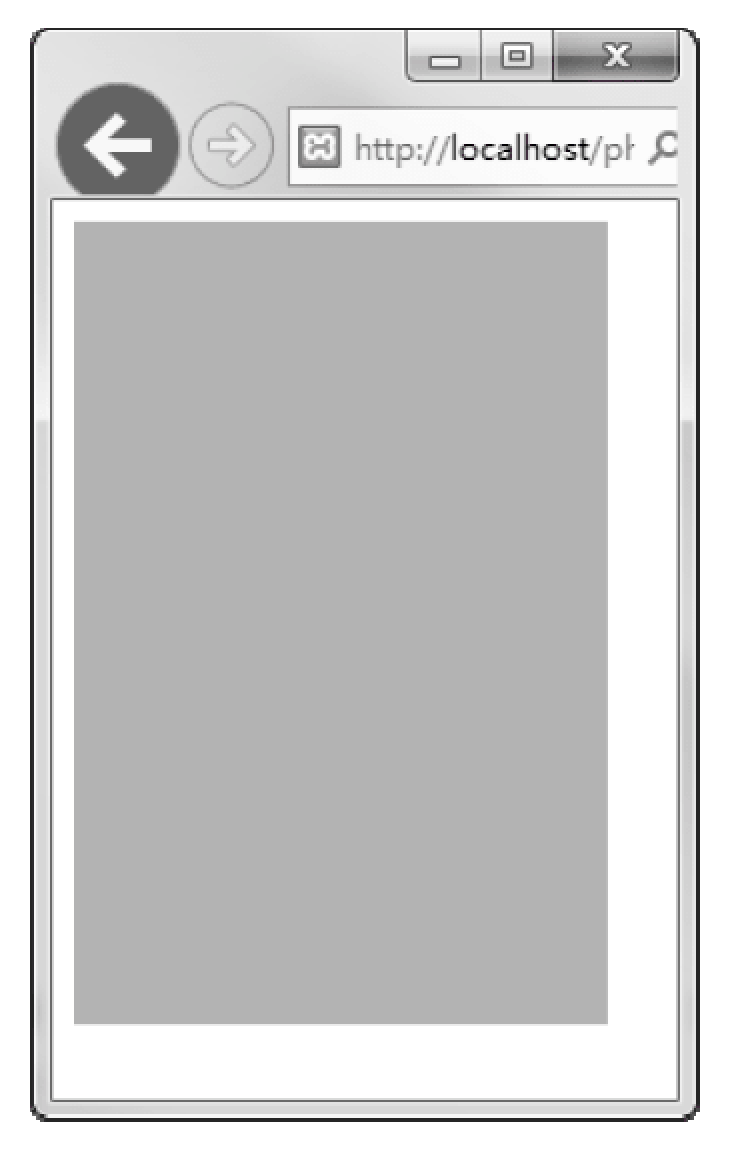


图 10.11　运行结果

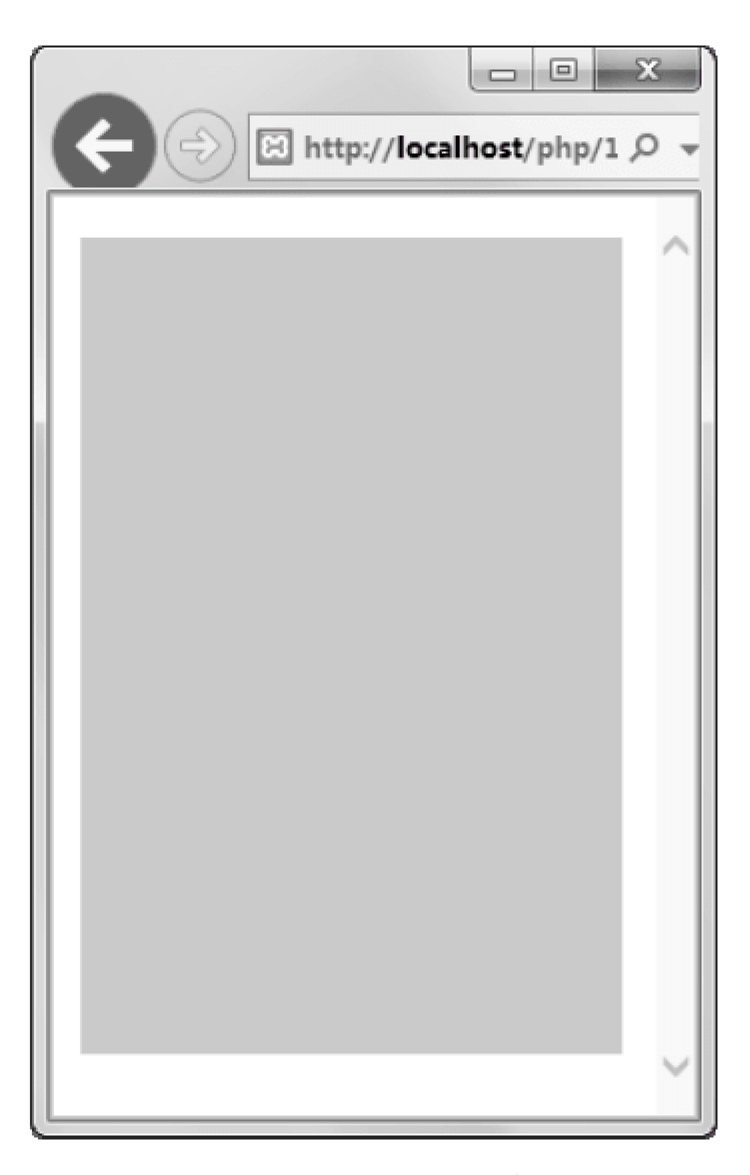


图 10.12　运行结果

为了便于演示效果，我们将透明度设置得比较高。从该示例的运行结果可以明显看到相对于不透明的黑色要浅很多。

10.2.3　获取图像信息

图像最直观的信息有宽和高。图像的其他信息主要有图像的类型及 Exif 信息。下面就来介绍获取这些信息的函数。

1. 获取图像的尺寸

获取图像尺寸即获取图像的宽和高，可以使用的函数有 3 个，它们的原型如下：

```
int imagesx ( resource $image )
int imagesy ( resource $image )
array getimagesize ( string $filename [, array &$imageinfo ] )
```

imagesx()和 imagesy()函数分别用于获取图像的宽度和高度，参数 image 为打开的图像资源。getimagesize()函数用来取得一个文件的大小以及一些其他的信息，结果将以数组的形式返回。参数 filename 为需要获取信息的文件名称，可选参数 imageinfo 用来获取图像的其他信息。

【示例 10-7】以下代码演示使用 imagesx()和 imagesy()函数获取图像的宽度和高度。

```
<?php
    $img=imagecreatefrompng('google_logo.png');      //打开图像文件
    $img_x=imagesx($img);                             //获取图像文件的宽度
    $img_y=imagesy($img);                             //获取图像文件的高度
    imagedestroy($img);                               //清除画布释放资源
    echo "该图像的宽度是{$img_x}px，高度是{$img_y}px。";//输出图像宽度和高度
?>
```

代码运行结果如图 10.13 所示。运行结果就输出了图像的大小信息。下面再来使用 getimagesize()函数获取图像文件的信息。

【示例 10-8】以下代码演示使用 getimagesize()函数获图像文件的信息。

```
<?php
    $img_info=getimagesize('google_logo.png'); //获取图像文件信息并返回
    echo '<pre>';                  //用于格式化显示后面的文字
    print_r($img_info);            //输出 getimagesize()函数返回的数组信息
?>
```

注意：getimagesize()函数会直接获取图像文件的信息，无须使用打开的文件资源。

代码运行结果如图 10.14 所示。

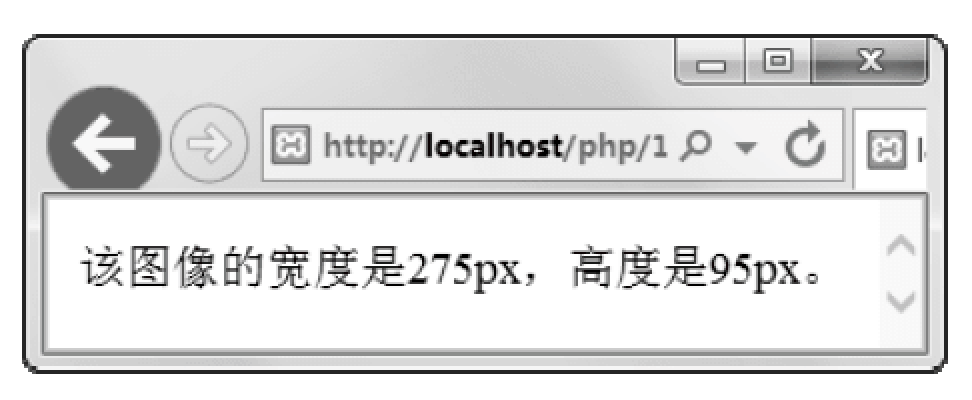


图 10.13　运行结果



图 10.14　运行结果

从运行结果中可以看到 getimagesize()函数返回的数组信息，我们依次来介绍各个数组元素表示的信息。

- 元素 0：表示图像文件的宽度。
- 元素 1：表示图像文件的高度。
- 元素 2：表示图像文件的格式。可能值为 1～16，主要的值有 1 为 GIF 格式（.gif）、2 为 JPEG/JPG 格式（.jpeg/.jpg）、3 为 PNG 格式（.png）。
- 元素 3：表示图像文件的宽度和高度，可以用于 HTML 标签的参数。
- 元素 bits：表示每种颜色的位数。
- 元素 mime：表示图像的 MIME 类型。

2. 检查图像是否为真彩色图像

检查图像是否为真彩色图像需要使用的函数是 imageistruecolor()，它的原型如下：

```
bool imageistruecolor ( resource $image )
```

参数 image 为打开的图像资源。该函数的使用和理解都比较简单这里就不详细介绍。

3. 获取图像的 Exif 信息

Exif 文件实际是 JPEG 文件的一种，遵从 JPEG 标准，只是在文件头信息中增加了有关拍摄信息的内容和索引图，包括拍摄时的光圈、快门、白平衡、ISO、焦距、日期时间等各种信息。现在很多网站中都会在展示的图像下方显示部分 Exif 信息。我们可以使用 exif_read_data()函数来获取这些信息，它的原型如下：

```
array exif_read_data ( string $filename [, string $sections [, bool $arrays
[, bool $thumbnail ]]] )
```

参数 filename 为需要获取 Exif 信息的文件名称；可选参数 sections 用于规定要返回的 Exif 信息，由于这些信息可能比较专业，因此，我们只使用默认选择即可；可选 arrays 用于规定结果是否以数组形式返回；可选参数 thumbnail 用于规定是否读取 Exif 信息中的缩略图，默认只读取标记数据。该函数可以读取 jpeg 和 tiff 格式的图像信息。

【示例 10-9】以下代码演示使用 exif_read_data()函数获取图像的相关信息。

```
<?php
    $img_info=exif_read_data('mypic.jpg');      //获取图像相关信息并返回
    echo '<pre>';                     //格式化后面的字符
    print_r($img_info);               //输出 exif_read_data 函数返回的数组信息
?>
```

代码运行结果如图 10.15 所示。

```
Array
(
    [FileName] => mypic.jpg
    [FileDateTime] => 1359611499
    [FileSize] => 872962
    [FileType] => 2
    [MimeType] => image/jpeg
    [SectionsFound] => ANY_TAG, IFD0, THUMBNAIL, EXIF, INTEROP
    [COMPUTED] => Array
        (
            [html] => width="2592" height="1456"
            [Height] => 1456
            [Width] => 2592
            [IsColor] => 1
            [ByteOrderMotorola] => 0
            [ApertureFNumber] => f/2.8
            [UserComment] =>
```

图 10.15　运行结果

由于篇幅的原因，这里只显示了该图片的一部分信息，我们可以看到该信息包含文件的名称、大小和类型等许多信息。

10.2.4　绘制图形

绘制图形是我们本章学习的重点，只要熟练掌握了绘制图形的知识，那么关于图像处理的其他知识都会很容易就掌握。当然绘制图形可能参数比较多，但是学习起来还是非常有趣的。

1. 绘制像素点

在图像处理中，PHP 可以处理的最小单位就是像素点。画一个像素点使用的函数是

imagesetpixel()函数，它的原型如下：

```
bool imagesetpixel ( resource $image , int $x , int $y , int $color )
```

参数 image 即为打开的图像资源；参数 x 和 y 分别为像素点的横坐标和纵坐标；参数 color 为像素点的颜色。为了使绘制效果更明显，我们会使用 imagefill()函数将背景填充为白色。

【示例 10-10】以下代码演示使用 imagesetpixel()函数绘制一个像素点。

```
<?php
    header("Content-type:image/png");                    //自定义报头
    $img=imagecreatetruecolor(200,200);                  //创建背景画布
    $black=imagecolorallocate($img,0,0,0);               //定义黑色
    $white=imagecolorallocate($img,255,255,255);         //定义白色
    imagefill($img,0,0,$white);                          //填充背景为白色
    imagesetpixel($img,100,100,$black);                  //绘制像素点
    imagepng($img);                                      //输出图像
    imagedestroy($img);                                  //清除图像释放资源
?>
```

代码运行结果如图 10.16 所示。在小学我们就学过点移动可以成线，下面我们就来演示使用像素点来画一条直线。

【示例 10-11】以下代码演示使用 for 循环绘制像素点组成一条直线。

```
<?php
    header("Content-type:image/png");                    //自定义报头
    $img=imagecreatetruecolor(200,200);                  //创建背景画布
    $black=imagecolorallocate($img,0,0,0);               //定义黑色
    $white=imagecolorallocate($img,255,255,255);         //定义白色
    imagefill($img,0,0,$white);                          //填充背景为白色
    for($i=0;$i<100;$i++)                                //定义一个循环
        imagesetpixel($img,$i,100,$black);               //绘制像素点
    imagepng($img);                                      //输出图像
    imagedestroy($img);                                  //清除图像释放资源
?>
```

代码运行结果如图 10.17 所示。

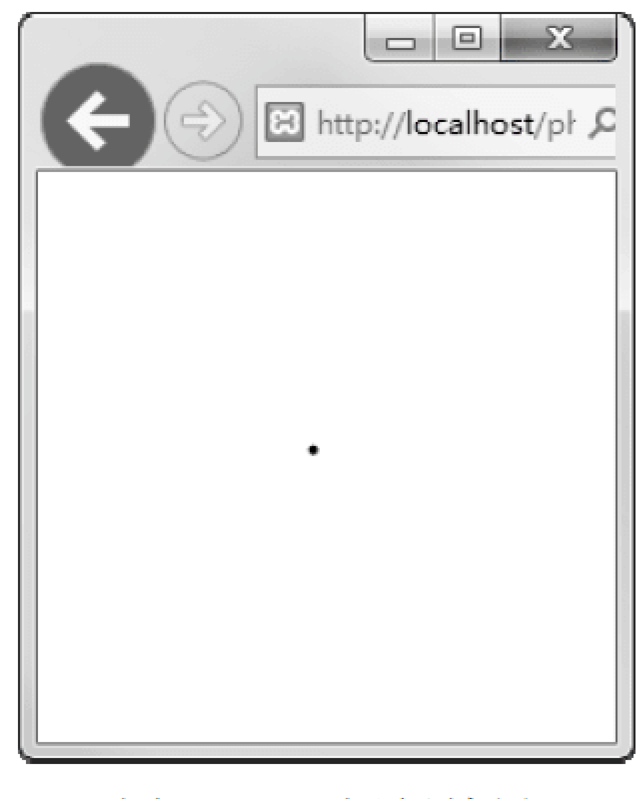


图 10.16　运行结果



图 10.17　运行结果

从图 10.17 所示的运行结果中可以很清楚地看到一条黑色的直线，这条直线就是使用 for 循环画出的像素点组成的。像素点的操作可以说是最灵活的，因为在屏幕上显示的任何图像都是由像素点组成的。下面我们就来演示画出一个黑色的矩形。

【示例 10-12】以下代码演示使用像素点画出一个黑色矩形。

```
<?php
    header("Content-type:image/png");                   //自定义报头
    $img=imagecreatetruecolor(200,200);                  //创建背景画布
    $black=imagecolorallocate($img,0,0,0);               //定义黑色
    $white=imagecolorallocate($img,255,255,255);         //定义白色
    imagefill($img,0,0,$white);                          //填充背景为白色
    for($i=0;$i<100;$i++){                               //纵坐标递增
        for($j=0;$j<100;$j++)                            //横坐标递增
            imagesetpixel($img,50+$j,50+$i,$black);//绘制黑色直线
    }
    imagepng($img);                                      //输出图像
    imagedestroy($img);                                  //清除图像释放资源
?>
```

代码运行结果如图 10.18 所示。图 10.18 中的黑色矩形就是使用像素点绘制的，当然绘制原理也很简单，循环绘制一条横线，然后再循环改变纵坐标。

2. 绘制直线

在示例 10-11 中我们使用像素点连起来画出了一条直线。在 PHP 中提供了专门的 imageline()函数来画出一条直线，该函数的原型如下：

```
bool imageline ( resource $image , int $x1 , int $y1 , int $x2 , int $y2 ,
int $color )
```

参数 image 为打开的图像资源；参数 x1 和 y1 分别为直线开始的横纵坐标；参数 x2 和 y2 分别为直线结束的横纵坐标；参数 color 为直线的颜色。

【示例 10-13】以下代码演示使用 imageline()函数绘制一条直线。

```
<?php
    header("Content-type:image/png");                   //自定义报头
    $img=imagecreatetruecolor(200,200);                  //创建背景画布
    $black=imagecolorallocate($img,0,0,0);               //定义黑色
    $white=imagecolorallocate($img,255,255,255);         //定义白色
    imagefill($img,0,0,$white);                          //填充背景为白色
    imageline($img,100,100,200,100,$black);              //绘制黑色直线
    imagepng($img);                                      //输出图像
    imagedestroy($img);                                  //清除图像释放资源
?>
```

代码运行结果如图 10.19 所示。

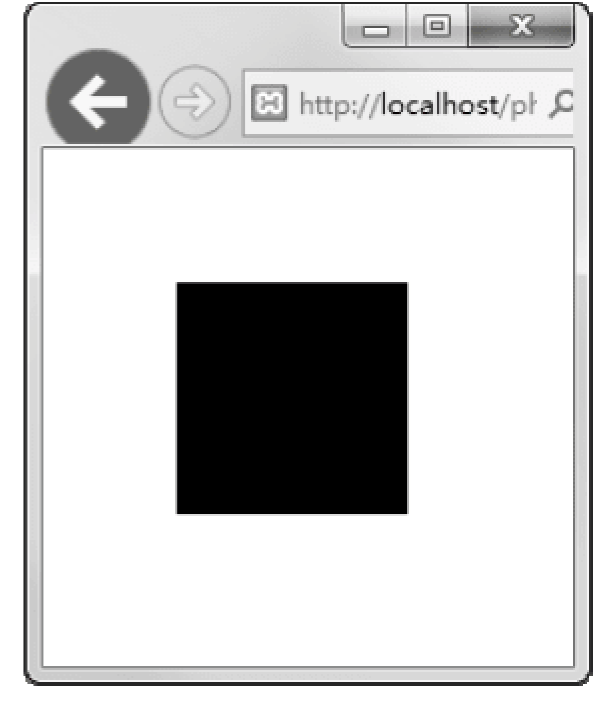


图 10.18　运行结果



图 10.19　运行结果

从运行结果中可以看到一条很明显的直线。当然，在一个画布中可以画不止一条直线，下面我们将会演示画出一个米字型的图像。

【示例 10-14】以下代码演示绘画出“米”字型图案。

```
<?php
    header("Content-type:image/png");                    //自定义报头
    $img=imagecreatetruecolor(200,200);                  //创建背景画布
    $black=imagecolorallocate($img,0,0,0);               //定义黑色
    $white=imagecolorallocate($img,255,255,255);         //定义白色
    imagefill($img,0,0,$white);                    //填充背景为白色
    imageline($img,0,0,200,200,$black);            //绘制左上角到右下角的直线
    imageline($img,0,200,200,0,$black);            //绘制左下角到右上角的直线
    imageline($img,0,100,200,100,$black);          //绘制横线
    imageline($img,100,0,100,200,$black);          //绘制竖线
    imagepng($img);                                //输出图像
    imagedestroy($img);                            //清除图像释放资源
?>
```

代码运行结果如图 10.20 所示。从运行结果中可以看到一个“米”字型的图案。该示例的重点就在于直线开始和结束位置坐标的确定。

3. 绘制矩形

绘制矩形可以使用 imagerectangle()函数实现，该函数的原型如下：

```
bool imagerectangle (resource $image, int $x1, int $y1, int $x2, int $y2,
int $col )
```

该函数通过指定将要绘制的矩形左上角和右下角坐标来绘制。参数 image 为打开的图像资源；参数 x1 和 y1 为矩形左上角的横纵坐标；参数 x2 和 y2 为矩形右下角的横纵坐标；参数 clo 为线条的颜色。

【示例 10-15】以下代码演示使用 imagerectangle()函数绘制一个矩形。

```
<?php
    header("Content-type:image/png");                    //自定义报头
    $img=imagecreatetruecolor(200,200);                  //创建背景画布
    $black=imagecolorallocate($img,0,0,0);               //定义黑色
    $white=imagecolorallocate($img,255,255,255);         //定义白色
    imagefill($img,0,0,$white);                          //填充背景为白色
    imagerectangle($img,50,50,150,100,$black);           //绘制矩形
    imagepng($img);                                      //输出图像
    imagedestroy($img);                                  //清除图像释放资源
?>
```

代码运行结果如图 10.21 所示。

从运行结果中可以看到该函数画出了一个矩形框。我们还可以使用 imagesetthickness()函数来设置画线的宽度，它的原型如下：

```
bool imagesetthickness ( resource $image , int $thickness )
```

参数 image 为打开的图像资源；参数 thickness 为需要设置的画线的宽度。

【示例 10-16】以下代码演示使用 imagesetthickness()函数设置画线的宽度并绘制一个

矩形。

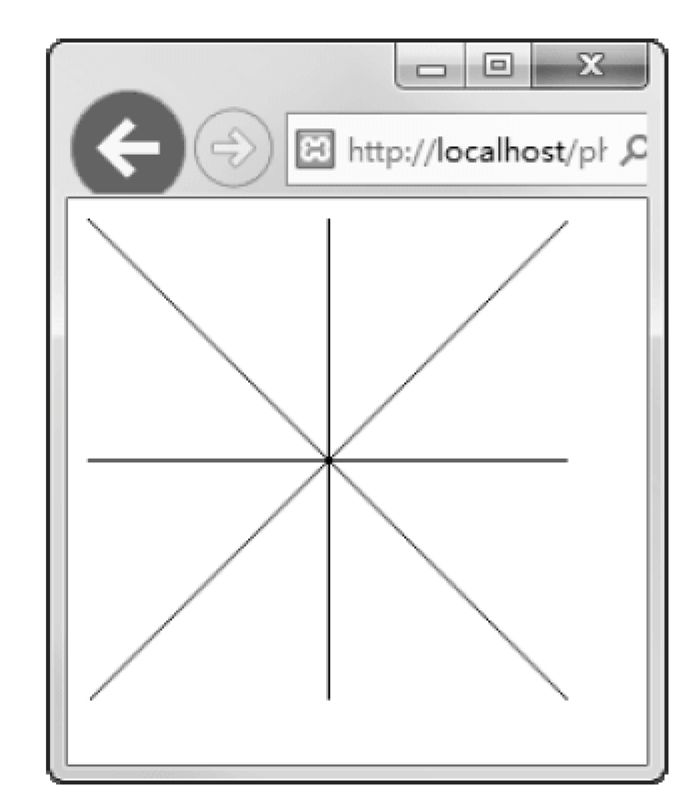


图 10.20　运行结果

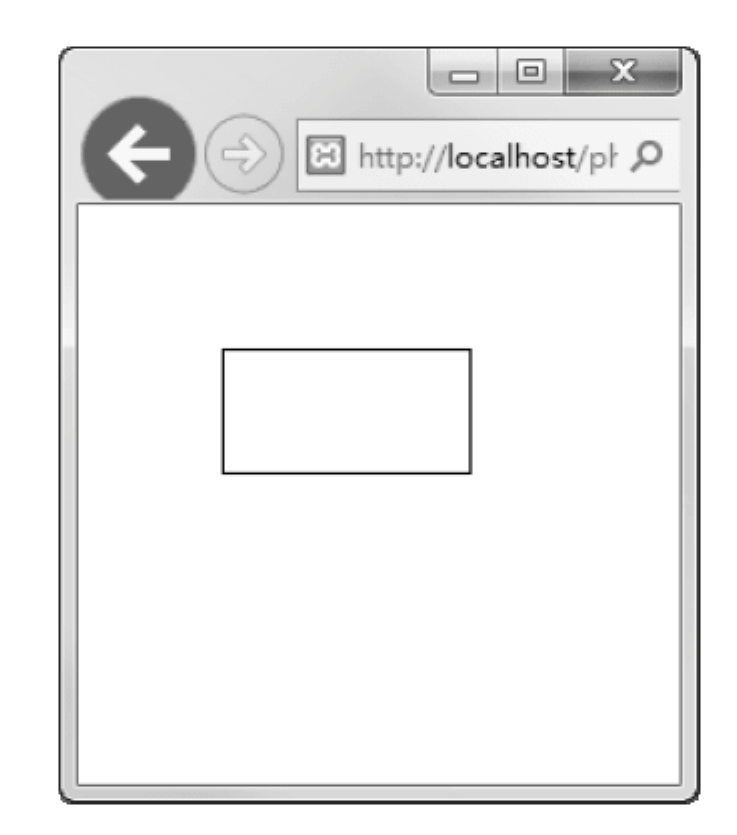


图 10.21　运行结果

```
<?php
    header("Content-type:image/png");                   //自定义报头
    $img=imagecreatetruecolor(200,200);                 //创建背景画布
    $black=imagecolorallocate($img,0,0,0);              //定义黑色
    $white=imagecolorallocate($img,255,255,255);        //定义白色
    imagefill($img,0,0,$white);                         //填充背景为白色
    imagesetthickness($img,3);                          //设置画线宽度为 3px
    imagerectangle($img,50,50,150,100,$black);          //绘制矩形
    imagepng($img);                                     //输出图像
    imagedestroy($img);                                 //清除图像释放资源
?>
```

代码运行结果如图 10.22 所示。从运行结果可以很明确看到画线的宽度增加了。在绘制好矩形框后，我们还可以使用 imagefilltoborder()函数来对矩形内部或者外部进行填充。imagefilltoborder()函数的原型如下：

```
bool imagefilltoborder ( resource $image , int $x , int $y , int $border ,
int $color )
```

参数 image 为打开的图像资源；参数 x 和 y 为填充开始的坐标点；参数 border 为填充的边界，即遇到边界则停止填充；参数 color 为填充的颜色。

【示例 10-17】以下代码演示使用 imagefilltoborder()函数填充矩形框内的区域。

```
<?php
    header("Content-type:image/png");                   //自定义报头
    $img=imagecreatetruecolor(200,200);                 //创建背景画布
    $black=imagecolorallocate($img,0,0,0);              //定义黑色
    $white=imagecolorallocate($img,255,255,255);        //定义白色
    imagefill($img,0,0,$white);                         //填充背景为白色
    imagerectangle($img,50,50,150,100,$black);          //绘制矩形
    imagefilltoborder($img,55,55,$black,$black);
                                                //填充以黑色为边框的区域
    imagepng($img);                                     //输出图像
    imagedestroy($img);                                 //清除图像释放资源
?>
```

代码运行结果如图 10.23 所示。

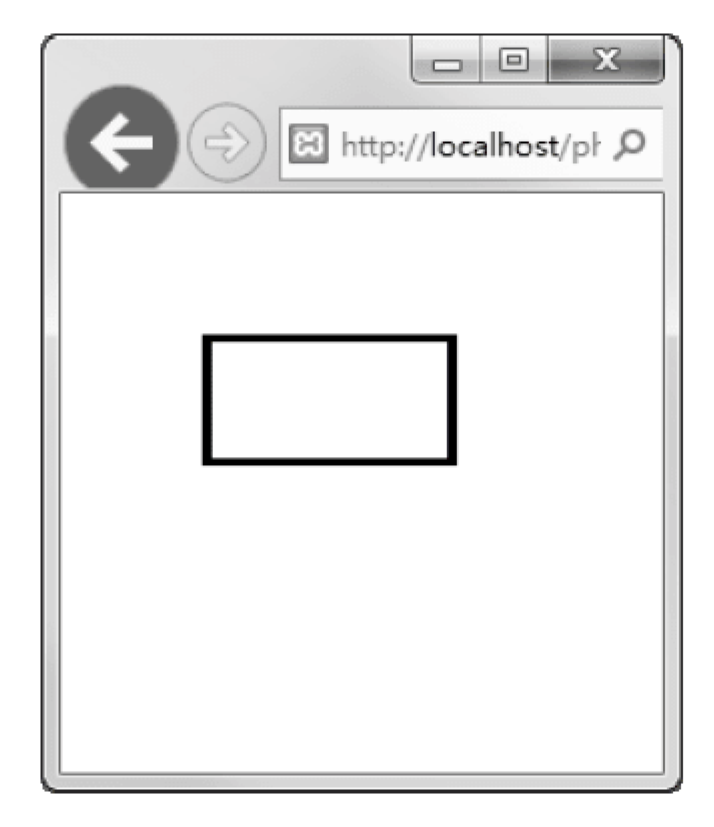
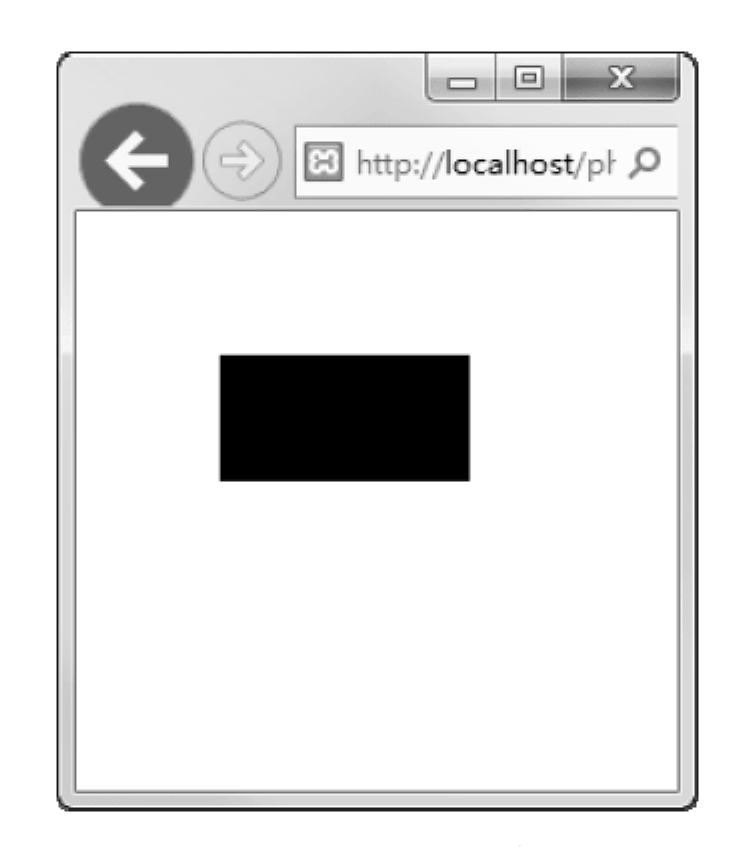



图 10.22　运行结果


图 10.23　运行结果

从运行结果可以看到，矩形区域框内被填充为黑色。在使用该函数的时候，只要将开始填充的坐标指定在矩形框内的任意一点即可。如果将开始填充的坐标点指定到矩形框外，则会填充矩形框外的区域。下面来演示使用 imagefilltoborder()函数填充两个矩形框之间的区域。

【示例 10-18】以下代码演示使用 imagefilltoborder()函数填充两个矩形框之间的区域。

```
<?php
    header("Content-type:image/png");                   //自定义报头
    $img=imagecreatetruecolor(200,200);                 //创建背景画布
    $black=imagecolorallocate($img,0,0,0);              //定义黑色
    $white=imagecolorallocate($img,255,255,255);        //定义白色
    imagefill($img,0,0,$white);                         //填充背景为白色
    imagerectangle($img,30,30,170,170,$black);          //绘制矩形
    imagerectangle($img,70,70,130,130,$black);          //绘制矩形
    imagefilltoborder($img,35,35,$black,$black);
                                                    //填充以黑色为边框的区域
    imagepng($img);                                     //输出图像
    imagedestroy($img);                                 //清除图像释放资源
?>
```

代码运行结果如图 10.24 所示。运行结果背景中黑色的区域即为 imagefilltoborder()函数填充的区域。其实在 PHP 中，还提供了绘制并填充矩形的函数 imagefilledrectangle()，该函数的原型如下：

```
bool imagefilledrectangle ( resource $image , int $x1 , int $y1 , int $x2 ,
int $y2 , int $color )
```

该函数的参数与 imagerectangle()函数非常类似，唯一的区别是参数 color 不再作为矩形边框的颜色而是作为填充的颜色。

【示例 10-19】以下代码演示使用 imagefilledrectangle()函数实现与示例 10-18 相同的运行效果。

```
<?php
    header("Content-type:image/png");                   //自定义报头
    $img=imagecreatetruecolor(200,200);                 //创建背景画布
    $white=imagecolorallocate($img,255,255,255);        //定义白色
    $black=imagecolorallocate($img,0,0,0);              //定义黑色
```

```
        imagefill($img,0,0,$white);                          //填充背景为白色
        imagefilledrectangle($img,30,30,170,170,$black);      //绘制矩形并填充
        imagefilledrectangle($img,70,70,130,130,$white);      //绘制矩形并填充
        imagepng($img);                                       //输出图像
        imagedestroy($img);                                  //清除图像释放资源
    ?>
```

代码运行结果如图 10.25 所示。

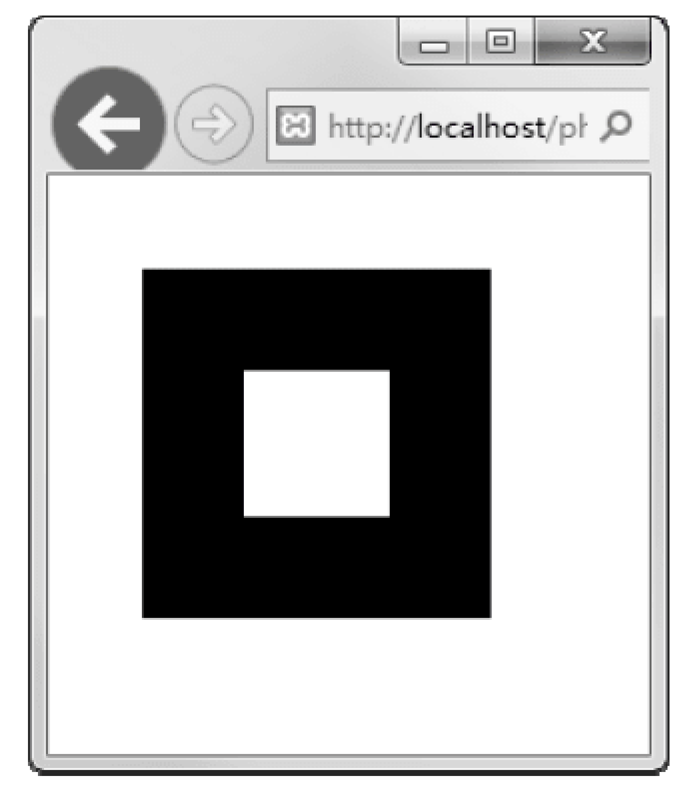


图 10.24　运行结果

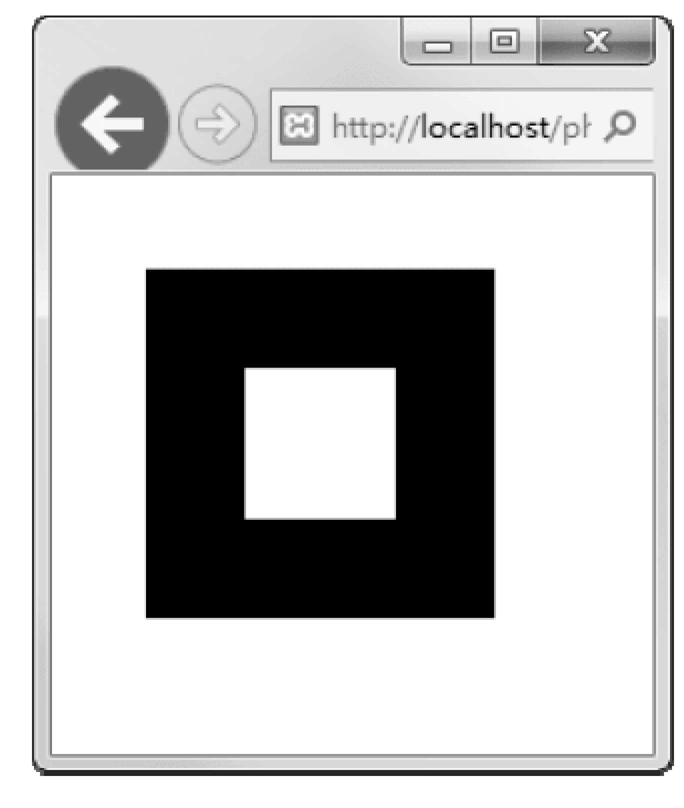


图 10.25　运行结果

我们可以看到该示例的运行结果与示例 10-18 的运行结果是完全相同的。

4. 绘制多边形

绘制多边形也有两个对应的 imagepolygon()和 imagefilledpolygon()函数，它们分别用来绘制一个多边形和绘制并填充多边形。imagepolygon()和 imagefilledpolygon()函数的原型如下：

```
bool imagepolygon ( resource $image , array $points , int $num_points , int
$color )
bool imagefilledpolygon ( resource $image , array $points , int $num_points ,
int $color )
```

参数 image 为打开的图像资源；参数 points 为一个数组，用来保存各个顶点的坐标，它的形式如下：

```
array(x1,y1,x2,y2,x3,y3,x4,…)
```

参数 numpoints 用来指定顶点的总数，必须要大于 3，numpoints 数组中多余的顶点会被省略。imagepolygon()函数的 color 参数为绘制的多边形线条的颜色；imagefilledpolygon()函数的 color 参数为多边形的填充色。

【示例 10-20】以下代码演示使用 imagepolygon()函数和 imagefilledpolygon()函数绘制多边形。

```
    <?php
        header("Content-type:image/png");                    //自定义报头
        $img=imagecreatetruecolor(200,200);                  //创建背景画布
        $black=imagecolorallocate($img,0,0,0);               //定义黑色
        $white=imagecolorallocate($img,255,255,255);         //定义白色
        imagefill($img,0,0,$white);                          //填充背景为白色
        $points=array(30,30,170,30,200,100,170,170,30,170,0,100);
```

```
        imagepolygon($img,$points,6,$black);              //绘制 6 边形
        imagefilledpolygon($img,$points,4,$black);        //绘制 4 边形并填充
        imagepng($img);                                   //输出图像
        imagedestroy($img);                               //清除图像释放资源
    ?>
```

代码运行结果如图 10.26 所示。在图 10.26 所示的运行结果中，六边形是使用 imagepolygon()函数绘制的，其中的黑色部分的四边形是使用 imagefilledpolygon()函数绘制并填充的。

5. 绘制椭圆

绘制椭圆可以使用 imageellipse()和 imagefilledellipse()函数完成，这两个函数的区别就是 imagefilledellipse()函数会对椭圆进行填充。这两个函数的原型如下：

```
bool imageellipse ( resource $image , int $cx , int $cy , int $w , int $h ,
int $color )
bool imagefilledellipse ( resource $image , int $cx , int $cy , int $w ,
int $h , int $color )
```

这两个函数的参数 image 为打开的文件资源；参数 cx 和 cy 为圆心的横纵坐标；参数 w 为椭圆的宽度；参数 h 为椭圆的高度。imageellipse 的 color 参数为线条的颜色；imagefilledellipse()函数的 color 参数为填充色。由于绘制椭圆的函数及操作都与绘制矩形类似，因此我们只以一个示例来演示。

【示例 10-21】以下代码演示使用 imageellipse()和 imagefilledellipse()函数绘制椭圆。

```
    <?php
        header("Content-type:image/png");                 //自定义报头
        $img=imagecreatetruecolor(200,200);               //创建背景画布
        $black=imagecolorallocate($img,0,0,0);            //定义黑色
        $white=imagecolorallocate($img,255,255,255);      //定义白色
        imagefill($img,0,0,$white);                       //填充背景为白色
        imageellipse($img,100,100,200,80,$black);         //绘制椭圆
        imagefilledellipse($img,100,100,80,200,$black);//绘制椭圆并填充
        imagepng($img);                                   //输出图像
        imagedestroy($img);                               //清除图像释放资源
    ?>
```

代码运行结果如图 10.27 所示。

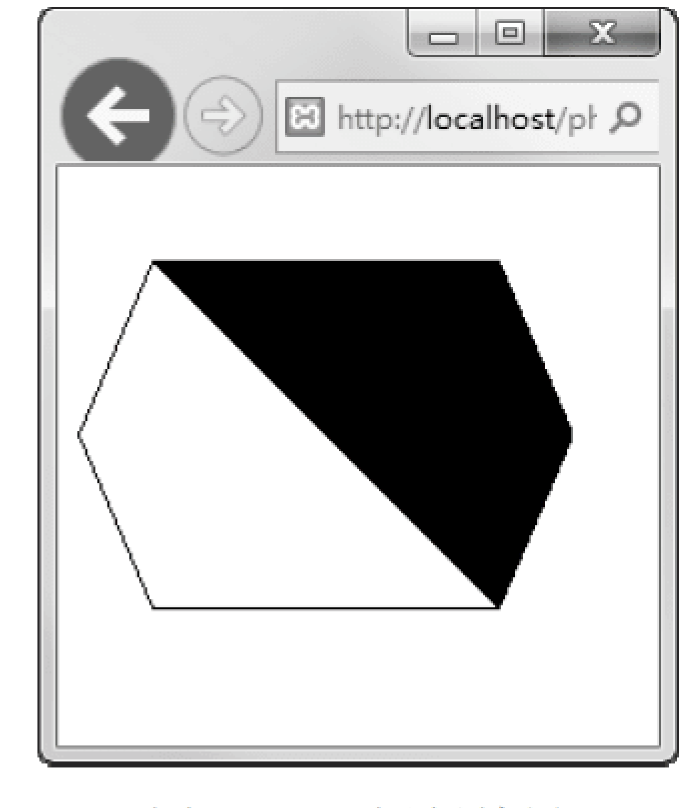


图 10.26　运行结果

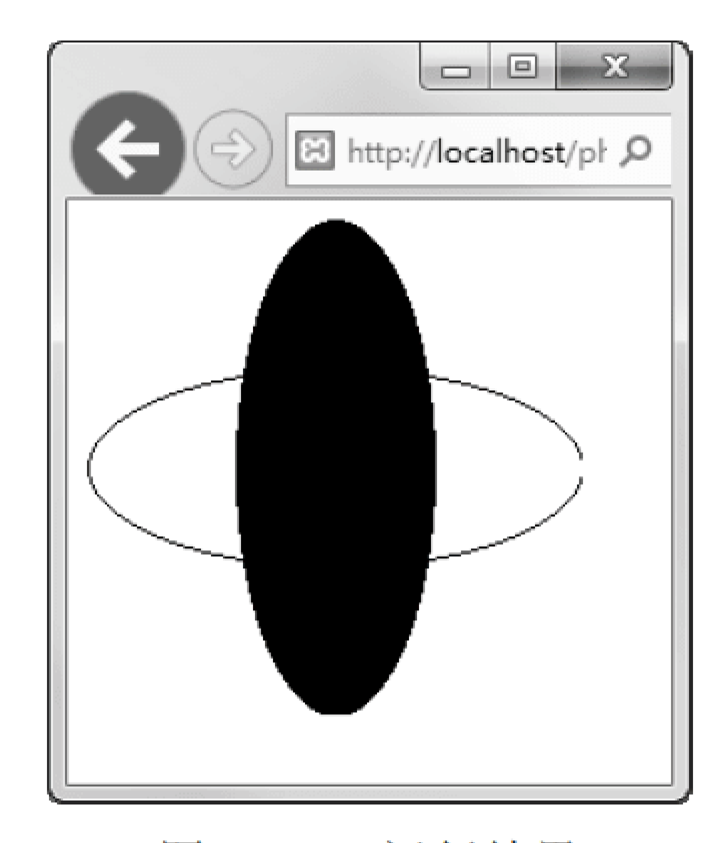


图 10.27　运行结果

运行结果中宽度较大的椭圆为使用 imageellipse()函数绘制的黑色线条椭圆；高度较大的椭圆为使用 imagefilledellipse()函数用黑色填充的椭圆。由于正圆是一种特殊的椭圆，因此使用 imageellipse()和 imagefilledellipse()函数同样可以用来绘制正圆。

6. 绘制弧线

绘制弧线对应的两个函数为 imagearc()和 imagefilledarc()函数。imagearc()函数用来绘制弧线，imagefilledarc()函数用来绘制弧线并填充。Imagearc()和 imagefilledarc()函数的原型如下：

```
bool imagearc ( resource $image , int $cx , int $cy , int $w , int $h , int
$s , int $e , int $color )
bool imagefilledarc ( resource $image , int $cx , int $cy , int $w , int
$h , int $s , int $e , int $color , int $style )
```

参数 image 即为打开的图像资源；参数 cx 和 cy 用来规定弧线的圆心；参数 w 用来规定弧线的宽度；参数 h 用来规定弧线的高度；参数 s 用来规定弧线的开始角度；参数 e 用来规定弧线的结束角度。imagearc()函数的 color 参数用来规定弧线的颜色。imagearc()函数用来规定对圆弧区域的填充色；参数 style 用来规定填充和绘制的风格，可选的参数及说明如下。

- IMG_ARC_PIE：产生圆形边界。
- IMG_ARC_CHORD：用直线连接起始和结束点并填充到圆心的区域。
- IMG_ARC_NOFILL：弧或弦只有轮廓，不填充。
- IMG_ARC_EDGED：直线将起始和结束点与中心点相连。

以上这些参数可以使用按位或组合起来实现多个效果。

【示例 10-22】以下代码演示使用 imagearc()函数绘制一条弧线。

```
<?php
    header("Content-type:image/png");                  //自定义报头
    $img=imagecreatetruecolor(200,200);                //创建背景画布
    $black=imagecolorallocate($img,0,0,0);             //定义黑色
    $white=imagecolorallocate($img,255,255,255);       //定义白色
    imagefill($img,0,0,$white);                        //填充背景为白色
    imagearc($img,100,100,100,150,30,330,$black);      //绘制弧线
    imagepng($img);                                    //输出图像
    imagedestroy($img);                                //清除图像释放资源
?>
```

代码运行结果如图 10.28 所示。

从运行结果中可以看到，使用 imagearc()函数绘制了一个类似于字母 C 的弧线。

【示例 10-23】以下代码演示使用 imagefilledarc()函数绘制并填充弧形区域。

```
<?php
    header("Content-type:image/png");                  //自定义报头
    $img=imagecreatetruecolor(400,400);                //创建背景画布
    $black=imagecolorallocate($img,0,0,0);             //定义黑色
    $white=imagecolorallocate($img,255,255,255);       //定义白色
    imagefill($img,0,0,$white);                        //填充背景为白色
    //绘制弧线并使用不同的填充风格
    imagefilledarc($img,100,100,180,150,30,330,$black,IMG_ARC_PIE);
    imagefilledarc($img,300,100,180,150,30,330,$black,IMG_ARC_CHORD);
    imagefilledarc($img,100,300,180,150,30,330,$black,IMG_ARC_NOFILL);
    imagefilledarc($img,300,300,180,150,30,330,$black,IMG_ARC_EDGED);
```

```
    imagepng($img);                                 //输出图像
    imagedestroy($img);                             //清除图像释放资源
?>
```

代码运行结果如图 10.29 所示。

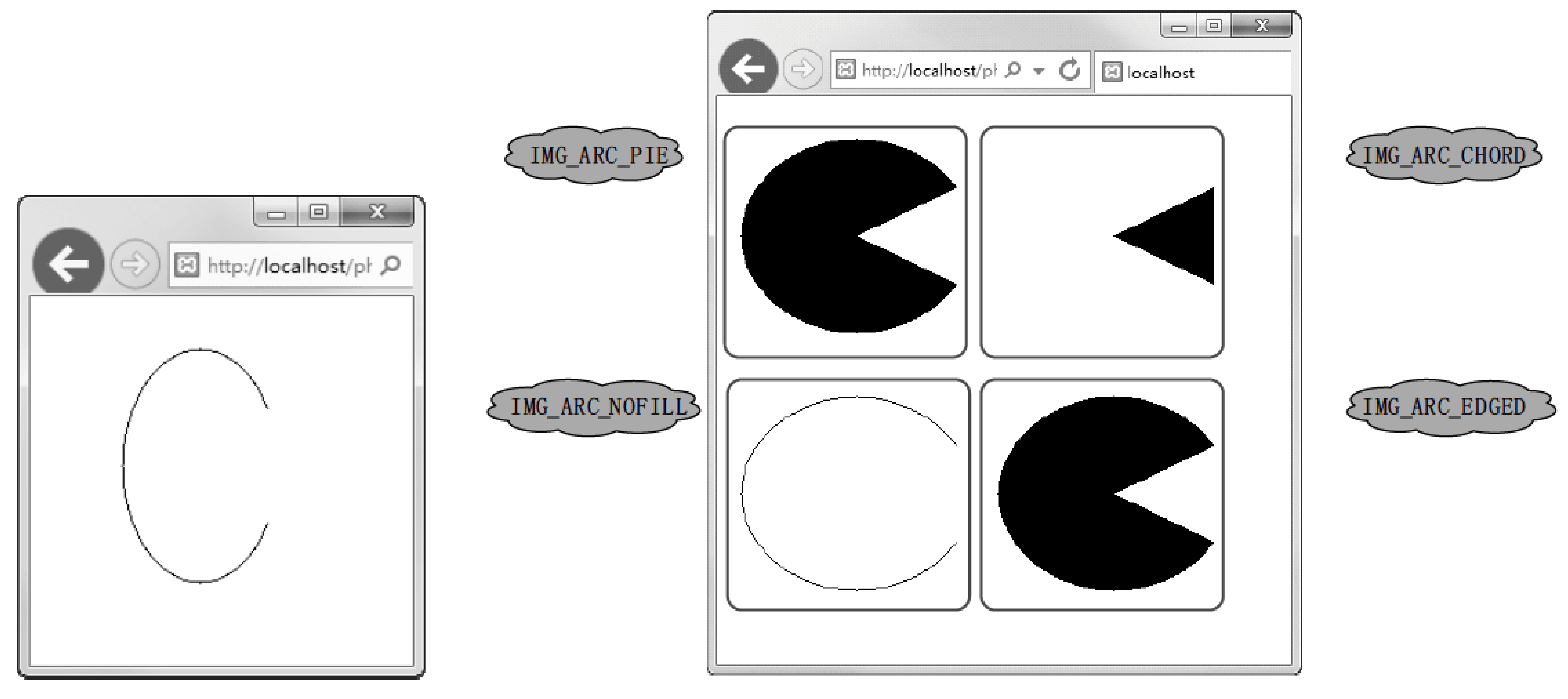


图 10.28　运行结果　　　　图 10.29　运行结果

从图 10.29 的运行结果中可以看到 4 种不同填充风格执行后的效果。从运行结果中看到使用 IMG_ARC_PIE 和 IMG_ARC_EDGED 执行的效果似乎是相同的，其实这是由于使用了填充而导致的，下面我们就使用逻辑或（|）来同时使用两种风格以去掉填充。

【示例 10-24】以下代码演示使用 IMG_ARC_NOFILL 去掉 IMG_ARC_PIE 和 IMG_ARC_EDGED 风格中的填充部分。

```
<?php
    header("Content-type:image/png");               //自定义报头
    $img=imagecreatetruecolor(400,400);             //创建背景画布
    $white=imagecolorallocate($img,255,255,255);    //定义白色
    //绘制弧线并使用不同的填充风格
    imagefilledarc($img,100,100,180,150,30,330,$white,
                    IMG_ARC_PIE|IMG_ARC_NOFILL);
    imagefilledarc($img,300,300,180,150,30,330,$white,
                    IMG_ARC_EDGED|IMG_ARC_NOFILL);
    imagepng($img);                                 //输出图像
    imagedestroy($img);                             //清除图像释放资源
?>
```

代码运行结果如图 10.30 所示。

从运行结果中可以很清楚地看到，它们绘制的轮廓线是不同的。下面来演示使用 imagefilledarc()函数画出具有 3D 效果的饼状图。

【示例 10-25】以下代码演示使用 imagefilledarc()函数绘制具有 3D 效果的饼状图。

```
<?php
    header("Content-type:image/png");               //自定义报头
    $img=imagecreatetruecolor(250,200);             //创建背景画布
    $color1=imagecolorallocate($img,0,155,150);     //定义底面颜色
    $color2=imagecolorallocate($img,0,35,255);      //定义表面颜色
```

```
    $white=imagecolorallocate($img,255,255,255);    //定义白色
    imagefill($img,0,0,$white);                      //填充背景为白色
    for($i=1;$i<8;$i++){                             //循环绘制圆弧并填充
        imagefilledarc($img,100,100+$i,180,150,30,330,$color1,
                            IMG_ARC_EDGED);
        imagefilledarc($img,120,100+$i,180,150,330,30,$color1,
                            IMG_ARC_EDGED);
    }
    //绘制表面
    imagefilledarc($img,120,100,180,150,330,30,$color2,
                        IMG_ARC_EDGED);
    imagefilledarc($img,100,100,180,150,30,330,$color2,
                        IMG_ARC_EDGED);
    imagepng($img);                                  //输出图像
    imagedestroy($img);                              //清除图像释放资源
?>
```

代码运行结果如图 10.31 所示。

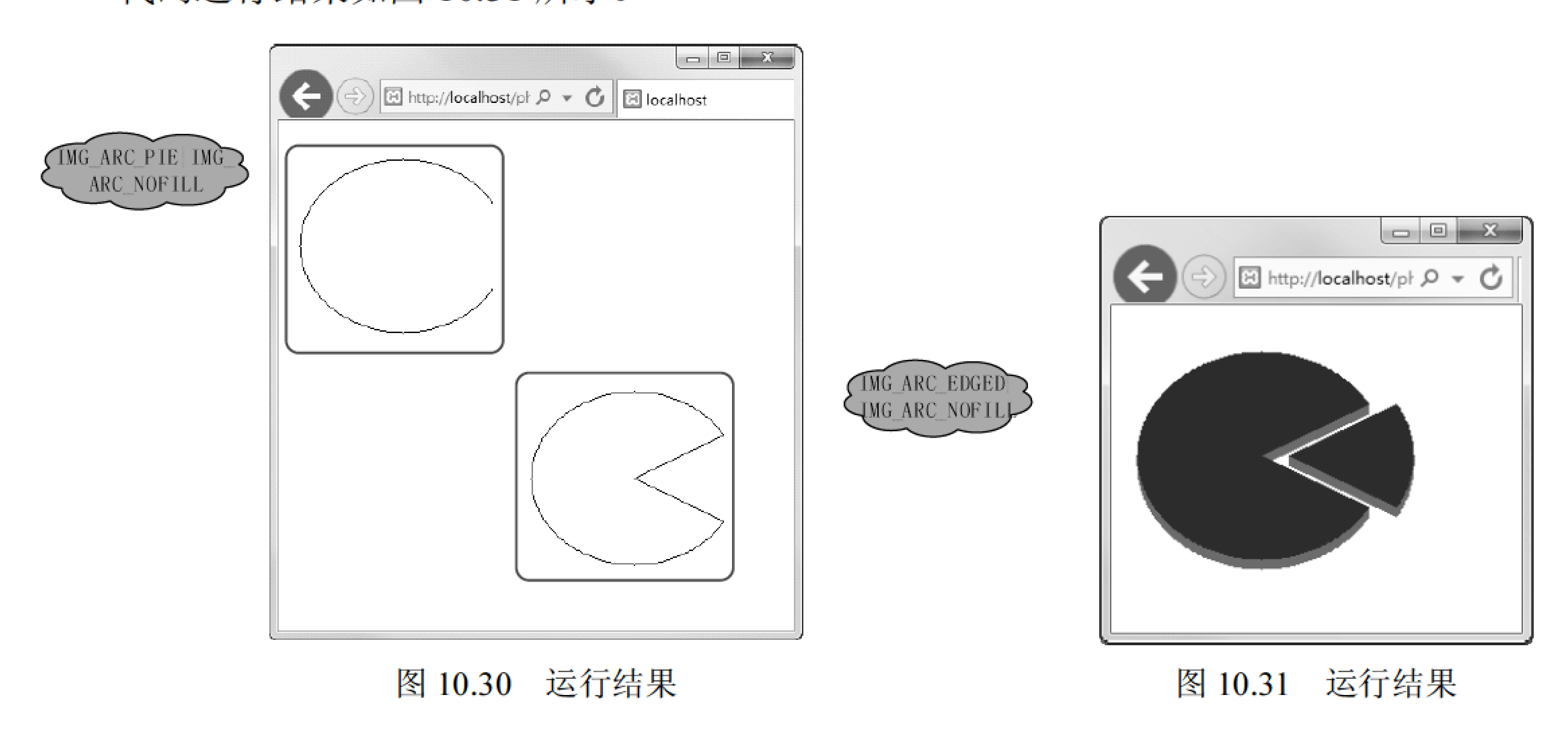


图 10.30　运行结果　　　　图 10.31　运行结果

从运行结果可以看到，程序通过循环简单实现了绘制具有 3D 效果的饼状图。到这里为止，绘制图形的知识就介绍完毕了。下面我们将介绍在图像上绘制文字的知识。

10.2.5　绘制文字

在图片上绘制文件最常见的应用就是验证码。下面我们就来简单学习在图像上绘制文字。

1. 绘制单个字符

绘制单个字符可以使用 imagechar()和 imagecharup()函数，这两个函数的原型如下：

```
bool imagechar ( resource $image , int $font , int $x , int $y , string $c ,
int $color )
bool imagecharup ( resource $image , int $font , int $x , int $y , string
$c , int $color )
```

参数 image 为打开的图像资源；参数 font 为要使用的字体，如果为数字 1～5 则使用内置字体，使用外部字体可以使用 imageloadfont()函数载入后使用；参数 x 和 y 为要绘制字符位置的横纵坐标；参数 c 为要绘制的字符，如果为字符串则只绘制第一个字符；参数

color 为字体的颜色。imagechar 将水平地绘制字符；imagecharup 将垂直地绘制字符。

【示例 10-26】以下代码演示使用 imagechar()和 imagecharup()函数绘制字符。

```
<?php
    header("Content-type:image/png");                    //自定义报头
    $img=imagecreatetruecolor(100,100);                  //创建背景画布
    $black=imagecolorallocate($img,0,0,0);               //定义黑色
    $white=imagecolorallocate($img,255,255,255);         //定义白色
    imagefill($img,0,0,$white);                          //填充背景为白色
    imagechar($img,5,55,10,'W',$black);                  //水平绘制字符 W
    imagecharup($img,5,20,60,'A',$black);                //垂直绘制字符 A
    imagepng($img);                                      //输出图像
    imagedestroy($img);                                  //清除图像释放资源
?>
```

代码运行结果如图 10.32 所示。可以从运行结果中看到，以水平方向绘制了字符 W 并以垂直方向绘制了字符 A。绘制字符的函数比较简单，这里就不做过多的介绍。

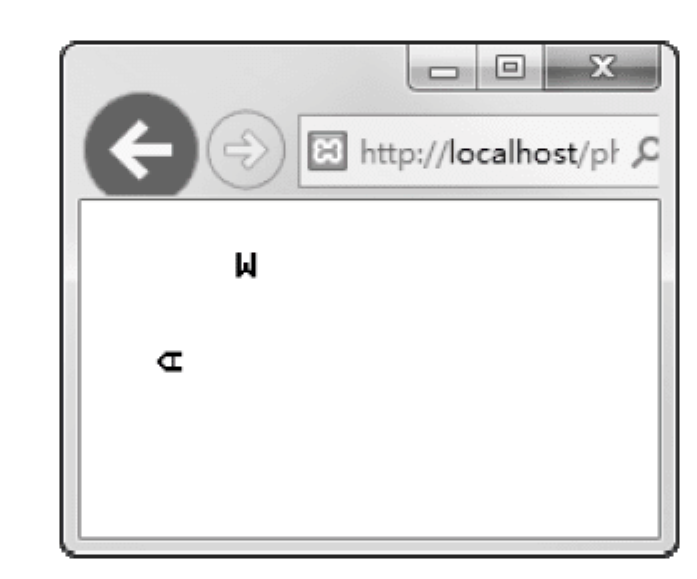


图 10.32　运行结果

2. 绘制字符串

在图像上绘制字符串使用的函数是 imagestring()和 imagestringup()函数。imagestring()函数会水平地绘制一行字符串，imagestringup()函数会垂直地绘制一行字符串。这两个函数的原型如下：

```
bool imagestring ( resource $image , int $font , int $x , int $y , string
$s , int $col )
bool imagestringup ( resource $image , int $font , int $x , int $y , string
$s , int $col )
```

imagestring()和 imagestringup()函数的参数说明与 imagechar()和 imagecharup()函数类似，这里就不再赘述。

【示例 10-27】以下代码演示使用 imagestring()和 imagestringup()函数绘制字符串。

```
<?php
    header("Content-type:image/png");                    //自定义报头
    $img=imagecreatetruecolor(200,200);                  //创建背景画布
    $black=imagecolorallocate($img,0,0,0);               //定义黑色
    $white=imagecolorallocate($img,255,255,255);         //定义白色
    imagefill($img,0,0,$white);                          //填充背景为白色
    $char='Draw my string!';                             //定义字符串
    imagestring($img,3,55,10,$char,$black);              //水平绘制字符串
    imagestringup($img,5,20,160,$char,$black);           //垂直绘制字符串
    imagepng($img);                                      //输出图像
    imagedestroy($img);                                  //清除图像释放资源
?>
```

代码运行结果如图 10.33 所示。

从运行结果可以看出，在图像中绘制了水平和垂直的两行字符串，由于水平方向的字符串使用 3 号字体，因此会比较小。

3. 绘制文本

绘制文本使用的函数是 imagettftext()函数，该函数绘制的文本又非常灵活，它的原型如下：

```
array imagettftext ( resource $image , float $size , float $angle , int $x ,
int $y , int $color , string $fontfile , string $text )
```

参数 image 为打开的图像资源；参数 size 为字体的大小；参数 angle 为文字的角度，参数会逆时针旋转文本，0 为从左向右；参数 x 和 y 为文本开始位置的横纵坐标会以字体的左下角为基本点；参数 color 为字体的颜色；参数 fontfile 为字体文件；参数 test 为需要绘制的文本。imagettftext()函数的绘制效果同 imagestring 等类似，下面演示在一幅图片上绘制字符串。

【示例 10-28】以下代码演示使用 imagettftext()函数在图像上绘制文字。

```
<?php
    header("Content-type:image/jpeg");                    //自定义报头
    $img=imagecreatefromjpeg('image.jpg');                //从文件创建背景
    $white=imagecolorallocate($img,255,255,255);          //定义白色
    $text='This is a beautiful flower.';                  //定义字符串
    //使用 imagettftext 函数绘制文本
    imagettftext($img,22,50,15,imagesy($img),$white,
                'C:\Windows\Fonts\courbd.ttf',$text);
    imagejpeg($img);                                      //输出图像
    imagedestroy($img);                                   //清除图像释放资源
?>
```

注意：在 Windows 操作系统中，字体文件通常在 C:\Windows\Fonts 路径下。

代码运行结果如图 10.34 所示。



图 10.33　运行结果



图 10.34　运行结果

从运行结果可以看出在以一朵花为背景的图像上我们在上面输出了“This is a beautiful flower.”字符串。

10.3　简易图片处理

PHP 除了可以绘制图形之外，还提供了简单的函数来实现一些常用的操作。例如对图片进行过滤、为图片添加水印等。

10.3.1　为图片添加水印

为了防止图片被未授权使用，我们可以通过对图片添加水印来保护图片。为图片添加水印的简单原理就是将事先只做好的水印图片复制到想要保护的图片中。在 PHP 中可以使用 imagecopy()函数来实现，该函数的原型如下：

```
bool imagecopy ( resource $dst_im , resource $src_im , int $dst_x , int $dst_y ,
int $src_x , int $src_y , int $src_w , int $src_h )
```

参数 dst_im 为源图像资源；参数 src_im 为需要复制到 dst_im 图片中的图像资源；参数 dst_x 和 dst_y 为复制进来的图像放置的坐标；参数 src_x、src_y、src_w 和 src_h 用来规定从 src_im 图像中复制从坐标(src_x,src_y)开始宽度为 src_w 高度为 src_h 的区域。该函数的参数比较多，读者在理解了该函数的作用后就可以正确使用这些参数了。

【示例 10-29】以下代码演示在 image.jpg 图像上添加 watermark.png 图像作为水印。

```
<?php
    header("Content-type:image/png");                    //自定义报头
    $img=imagecreatefromjpeg('image.jpg');               //打开源文件
    $watermark=imagecreatefrompng('watermark.png');//打开水印文件
    //使用 imagecopy 将 watermark 图像复制到 image 图像的右上角
    imagecopy($img,$watermark,imagesx($img)-imagesx($watermark),
              0,0,0,imagesx($watermark),imagesy($watermark));
    imagepng($img);                                      //输出图像
    imagedestroy($img);                                  //清除图像释放资源
?>
```

代码运行结果如图 10.35 所示。



图 10.35　运行结果

从图 10.35 所示图中的“请勿盗用”即为我们添加的水印图片。由于我们着重介绍的是该函数，因此水印做得比较粗糙，通常情况，水印会有透明效果，以防止影响源图片美观程度。同时由于我们不确定水印图片的大小，因此在为 imagecopy()函数赋值的时候使用了一些计算。

10.3.2　对相片使用过滤器

PHP 使用 imagefilter()函数来对相片使用过滤器。所谓过滤器就类似于滤镜，可以很容易地实现一些效果。imagefilter()函数的原型如下：

```
bool imagefilter ( resource $src_im , int $filtertype [, int $arg1 [, int
$arg2 [, int $arg3 ]]] )
```

参数 src_im 为打开的图像资源；参数 filtertype 用来规定过滤的类型，可以使用的参数及说明如下：

- IMG_FILTER_NEGATE：将图像中所有颜色反转。
- IMG_FILTER_GRAYSCALE：将图像转换为灰度的。
- IMG_FILTER_BRIGHTNESS：通过使用 arg1 设定亮度级别改变图像的亮度。
- IMG_FILTER_CONTRAST：改变图像的对比度。用 arg1 设定对比度级别。
- IMG_FILTER_COLORIZE：通过使用 arg1，arg2 和 arg3 分别指定 red，blue 和 green 来转换图像。
- IMG_FILTER_EDGEDETECT：用边缘检测来突出图像的边缘。
- IMG_FILTER_EMBOSS：使图像浮雕化。
- IMG_FILTER_GAUSSIAN_BLUR：用高斯算法模糊图像。
- IMG_FILTER_SELECTIVE_BLUR：模糊图像。
- IMG_FILTER_MEAN_REMOVAL：用平均移除法来达到轮廓效果。
- IMG_FILTER_SMOOTH：通过使用 arg1 设定柔滑级别使图像更柔滑。

该函数虽然使用简单，但是实现的效果还是比较理想的，下面就来演示其中一些效果。

【示例 10-30】以下代码演示使用 imagefilter()函数对图像使用过滤器。

```
<?php
    header("Content-type:image/jpeg");      //自定义报头
    $src1=imagecreatefromjpeg('image.jpg');//打开源文件
    imagefilter($src1,IMG_FILTER_NEGATE);  //对图片颜色进行反转处理
    $src2=imagecreatefromjpeg('image.jpg');//打开源文件
    imagefilter($src2,IMG_FILTER_EMBOSS);  //对图像进行浮雕化处理
    $src3=imagecreatefromjpeg('image.jpg');//打开源文件
    imagefilter($src3,IMG_FILTER_MEAN_REMOVAL);
                                   //对文件进行增强轮廓效果处理
    $img_w=imagesx($src1);         //获取源图像宽度
    $img_h=imagesy($src1);         //获取源图像高度
    $img=imagecreatetruecolor($img_w*3,$img_h);
                                   //创建三倍于 image.jpg 图像宽度的画布
    //将 image.jpg 复制三份到背景中
    imagecopy($img,$src1,0,0,0,0,$img_w,$img_h);
    imagecopy($img,$src2,$img_w,0,0,0,$img_w,$img_h);
    imagecopy($img,$src3,$img_w*2,0,0,0,$img_w,$img_h);
    imagejpeg($img);               //输出图像
    imagedestroy($img);            //清除图像释放资源
?>
```

代码运行结果如图 10.36 所示。

从运行结果我们可以看到，相同的原图片经过不同风格的处理都取得了比较理想的效果。当然这里只演示了其中 3 种效果，读者可以自己尝试其他剩余未演示的效果。

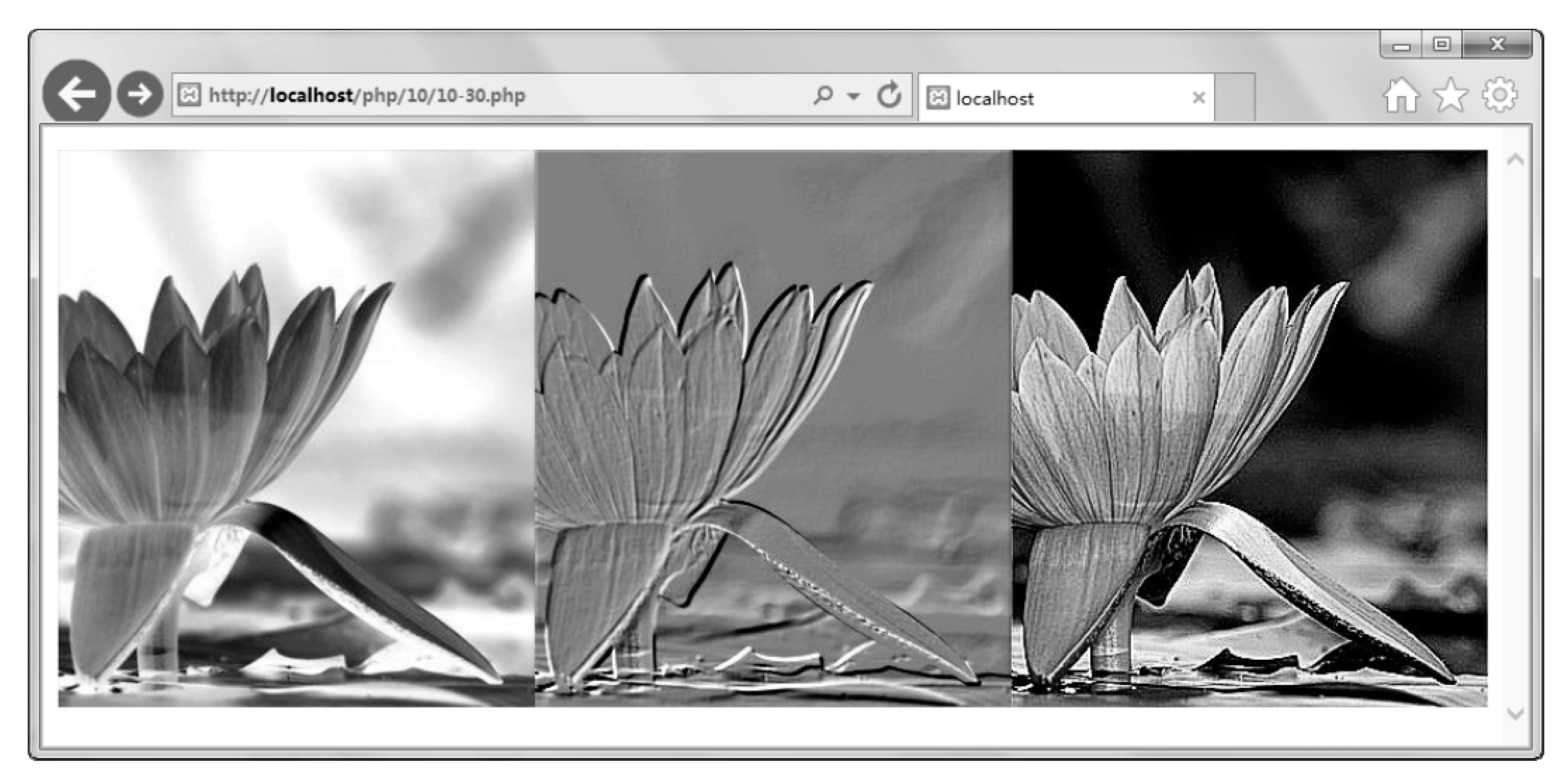


图 10.36　运行结果

10.4　生成验证码

生成验证码可以说是我们前面学习的所有知识的一个综合应用。因为生成验证码首先要在图片中绘制文件，其次要加入一些干扰元素，我们可以随机生成一些像素点、直线、椭圆之类的图形。下面我们将会简单实现一个验证码图片。

首先我们会在验证码生成一些随机的数字或者字母。首先，生成随机数字我们可以使用 rand 函数来生成，它的原型如下：

```
int rand ([ int $min ], int $max )
```

可选参数 min 用来设置生成的随机数的最小值；参数 max 用来设置生成的随机数的最大值。在有了可以生成指定范围的随机数的函数后，那么随机生成大小写字母的问题也就迎刃而解了。我们可以使用以下形式来生成大小写字母：

```
sprintf('%c',rand(65,90))                  //将数字转换为大写字母
sprintf('%c',rand(97,122))                 //将数组转换为小写字母
```

那么下面我们首先来实现随机产生字符。

【示例 10-31】以下代码演示生成 4 位随机字符串。

```
<?php
    $str='';                               //定义空字符用来接收字符串
    for($i=0;$i<4;$i++){                   //使用 for 循环生成 4 位随机字符
        switch(rand(1,3)){                 //使用 switch 随机选择生成的字符
            case 1:
                $str.=sprintf('%c',rand(65,90));
                                           //生成随机大写字母并加入到$str 变量中
                break;
            case 2:
                $str.=sprintf('%c',rand(97,122));
                                           //生成随机小写字母并加入到$str 变量中
                break;
            case 3:
```

```
                $str.=rand(0,9);     //生成随机数字并加入到$str 变量中
                break;
            }
        }
        echo $str;                   //输出生成的随机字符串
    ?>
```

代码运行结果如图 10.37 所示。我们可以看到 4 位随机字符串已经产生了，当然读者运行该示例的时候，运行结果可能各不相同，因为这个结果是随机的。我们接着来实现随机生成一些图形干扰元素。



图 10.37　运行结果

首先，实现随机像素点可以用如下的代码：

```
for($i=0;$i<100;$i++)
    imagesetpixel($img,rand(0,$img_x),rand(0,$img_y),$white)
```

实现随机直线也可以使用类似的代码：

```
for($i=0;$i<100;$i++)
    imageline($img,rand(0,$img_x),rand(0,$img_y),rand(0,$img_x),rand(0,$
img_y),$white)
```

当然我们还可以加入其他一些图形，例如圆形、圆弧和多边形等，它们的实现方法都类似。这里我们就不做详细介绍了，读者可以自己发挥。

【示例 10-32】以下代码演示生成随机像素点和直线。

```
<?php
    header("Content-type:image/png");                 //自定义报头
    $img_x=200;                                       //定义画布宽度
    $img_y=100;                                       //定义画布高度
    $img=imagecreatetruecolor($img_x,$img_y);         //创建背景画布
    $white=imagecolorallocate($img,255,255,255);      //定义白色
    //循环输出随机线条
    for($i=0;$i<10;$i++)
        imageline($img,rand(0,$img_x),rand(0,$img_y),rand(0,$img_x),
                 rand(0,$img_y),$white);
    //循环输出随机像素点
    for($i=0;$i<100;$i++)
        imagesetpixel($img,rand(0,$img_x),rand(0,$img_y),$white);
    imagepng($img);                                   //输出图像
    imagedestroy($img);                               //清除图像释放资源
?>
```

代码运行结果如图 10.38 所示。

我们可以看到生成的干扰元素还是比较理想的，下面我们就将文字和干扰元素整合起来。

【示例 10-33】以下代码演示生成验证码。

```
<?php
    header("Content-type:image/png");   //自定义报头
    $str='';                            //定义空字符用来接收字符串
    $img_x=200;                         //定义画布宽度
    $img_y=100;                         //定义画布高度
    $img=imagecreatetruecolor($img_x,$img_y);         //创建背景画布
    $white=imagecolorallocate($img,255,255,255);      //定义白色
```

```
    //循环输出随机线条
    for($i=0;$i<10;$i++)
        imageline($img,rand(0,$img_x),rand(0,$img_y),rand(0,$img_x),
                  rand(0,$img_y),$white);
    //循环输出随机像素点
    for($i=0;$i<100;$i++)
        imagesetpixel($img,rand(0,$img_x),rand(0,$img_y),$white);
    for($i=0;$i<4;$i++){                //使用 for 循环生成 4 位随机字符
        switch(rand(1,3)){              //使用 switch 随机选择生成的字符
            case 1:
                $str.=sprintf('%c',rand(65,90));
                                        //生成随机大写字母并加入到$str 变量中
                break;
            case 2:
                $str.=sprintf('%c',rand(97,122));
                                        //生成随机小写字母并加入到$str 变量中
                break;
            case 3:
                $str.=rand(0,9);        //生成随机数字并加入到$str 变量中
                break;
        }
    }
    //将随机字符串写入画布并使用一个小幅随机的角度
    imagettftext($img,45,rand(0,15),25,75,$white,
                 'C:\Windows\Fonts\courbd.ttf',$str);
    imagepng($img);                     //输出图像
    imagedestroy($img);                 //清除图像释放资源
?>
```

代码运行结果如图 10.39 所示。

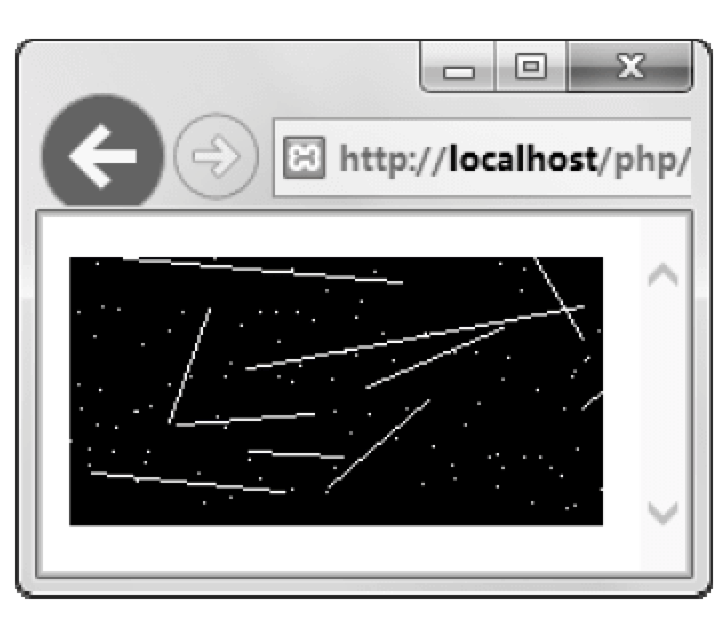


图 10.38　运行结果



图 10.39　运行结果

从运行结果可以看到，生成的验证码还是比较理想的，读者也可以修改示例中的代码或者使用更好的方法来生成。

10.5　小　　结

本章主要介绍了如何使用 PHP 中的函数绘制各种图形。同时还简单介绍了两种处理图片的方法。最后我们使用利用前面所有的知识完成了验证码的生成。本章最重点的部分就在于图像的坐标系统。经过许多示例的演示，相信读者已经能够熟练地使用这些函数来绘制出期望的图形。

10.6　本 章 习 题

1. 使用 header()函数指定文件的 MIME 类型为 image/png。
2. 使用 imagecreatetruecolor()函数创建一张真彩色画布。
3. 使用 imagecolorallocate()函数定义青色并填充习题 2 创建的画布。
4. 使用 imagefilledellipse()函数绘制一个圆形并使用青色填充。

第 11 章　数据库管理系统

数据库管理系统可以说是现在 Web 应用中必不可少的一部分。现在的数据库管理系统有很多，常见的有 MySQL、Oracle、SQL Server 和 PostgreSQL 等。常与 PHP 搭配使用的数据库管理系统是 MySQL。流行的 LAMP 组合就是 Linux、Apache、MySQL 和 PHP。由此也可以反映出 MySQL 和 PHP 搭配的流行程度。本章我们就主要介绍使用 PHP 操作 MySQL。

11.1　MySQL 基础

虽然 PHP 和 MySQL 组合是非常流行的，但它们毕竟是两种不同的技术。因此现在我们需要暂缓 PHP 的学习，来了解一些 MySQL 或者说是对大多数数据库的操作基础。

11.1.1　使用 MySQL 数据库前的准备

由于我们使用的是集成环境，因此相关配置已经设置得比较通用了，但是在默认的情况下通常需要配置 php.ini 文件，来获取 PHP 对 MySQL 的支持。php.ini 文件中需要设置的选项如下：

```
extension=php_mysql.dll
extension=php_mysqli.dll
```

在确认了以上模块被加载之后，我们就可以从 XAMPP 控制面板中启动 MySQL 服务，启动后的状态如图 11.1 所示。

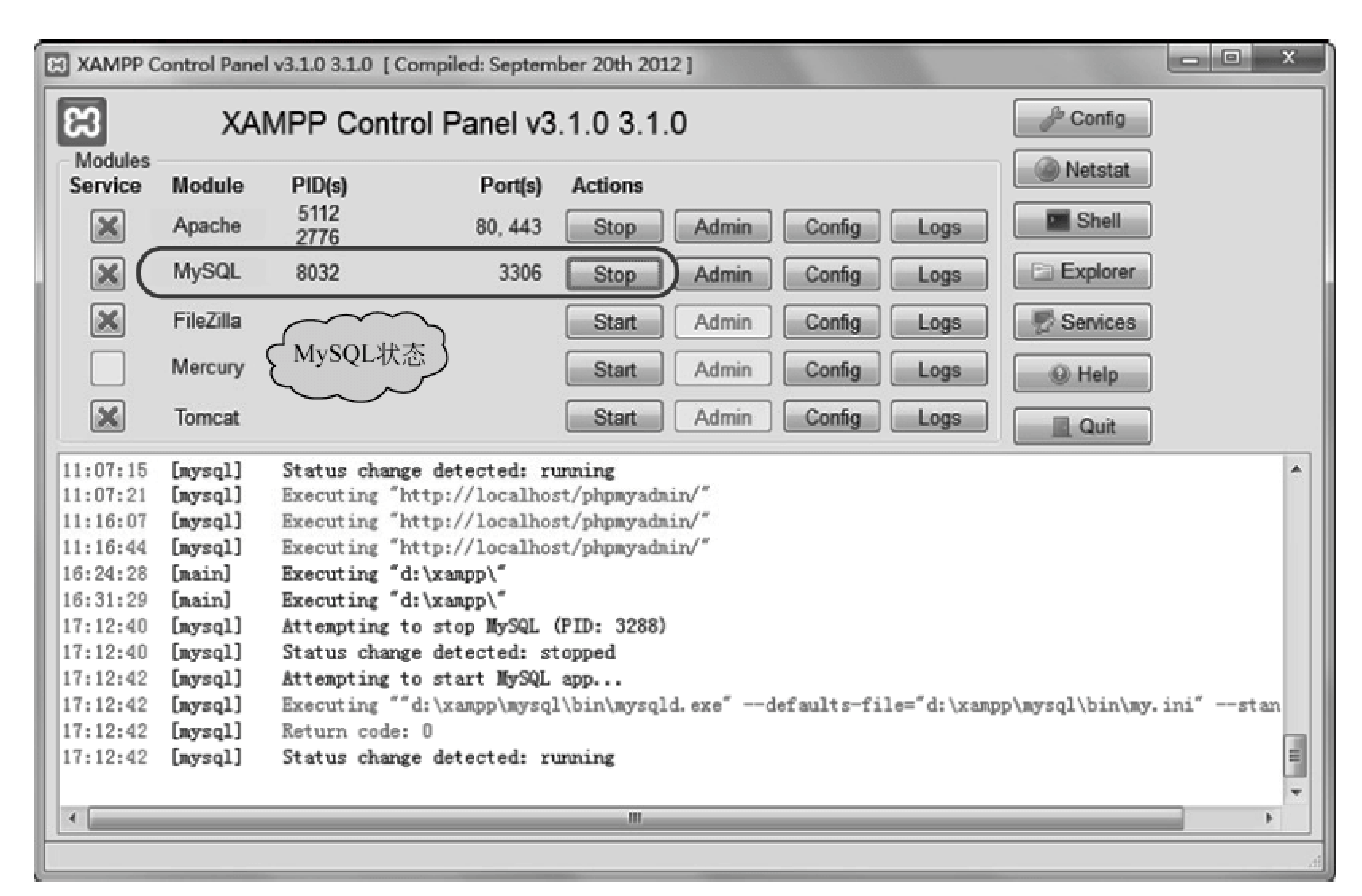


图 11.1　MySQL 启动成功后的状态

在 MySQL 成功启动后，就为后续操作做好了基础。

11.1.2　连接与断开 MySQL 数据库

数据库管理系统是用来管理数据的。同操作文件类似，要对数据进行操作，就需要打开一个到数据的连接。MySQL 默认的客户端是基于命令行的，我们可以通过 XAMPP 控制面板中的 Shell 按钮来打开一个命令行窗口，如图 11.2 所示。

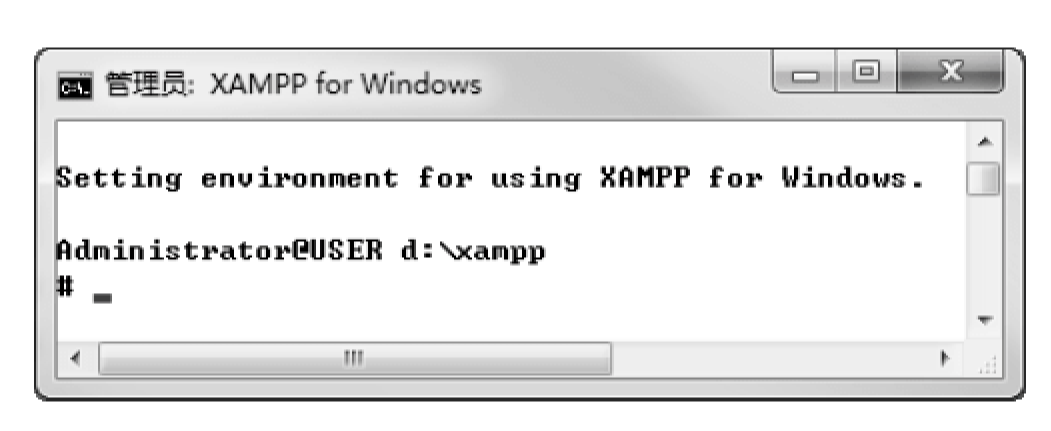


图 11.2　命令行窗口

我们可以在命令行中输入如下的命令来打开与 MySQL 的连接：

```
mysql -u 用户名 -p 密码
```

在 XAMPP 集成环境中，默认的用户名为 root，密码为空。在把上述命令中的“用户名”替换为“root”按下 Enter 键提交后会提示我们输入密码，由于密码为空，直接回车即可。然后窗口中就会显示 MySQL 的欢迎信息，如图 11.3 所示。

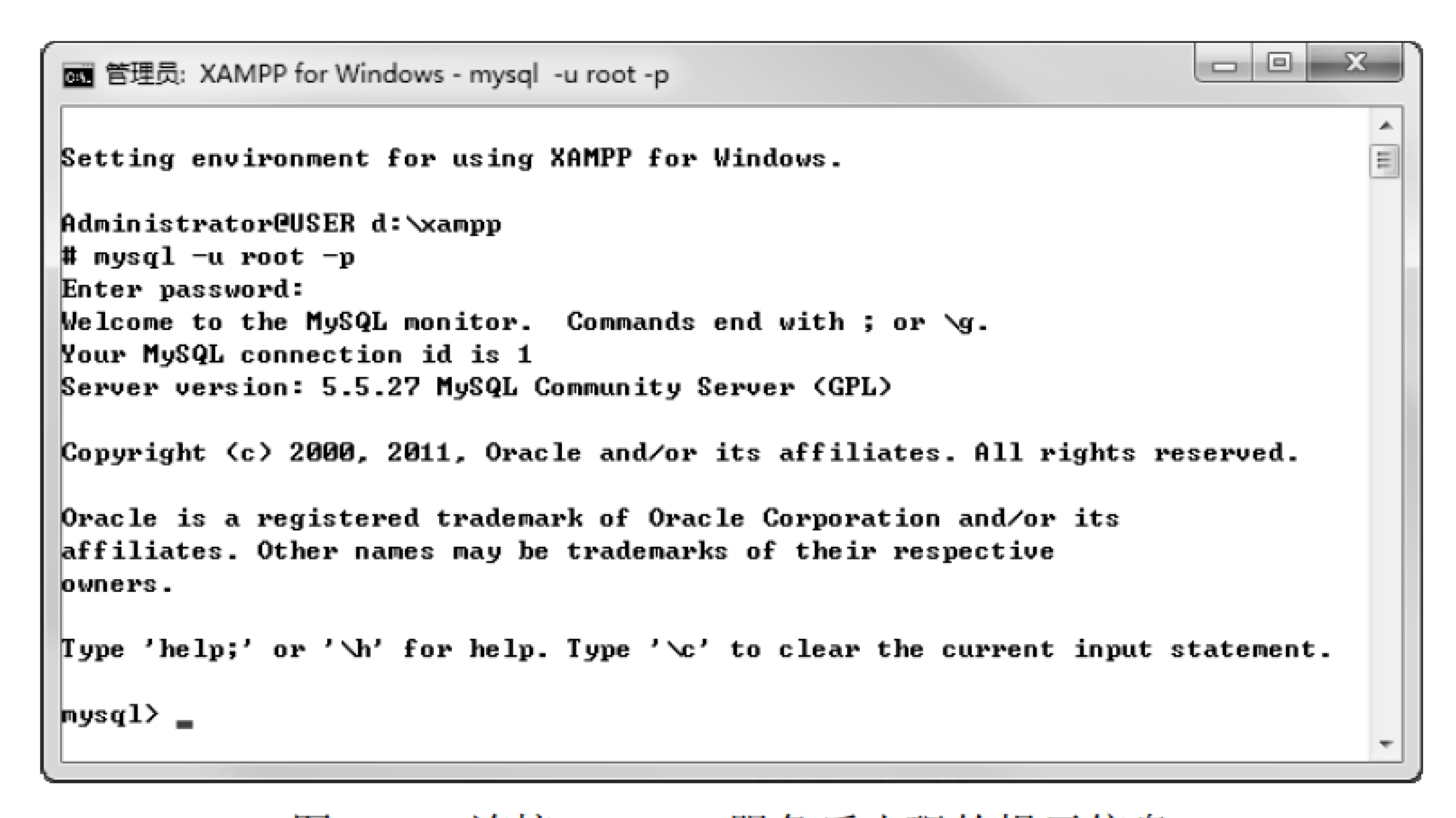


图 11.3　连接 MySQL 服务后出现的提示信息

现在我们就已经连接了 MySQL。断开与 MySQL 的连接可以输入 exit 命令来退出，退出后的窗口状态如图 11.4 所示。

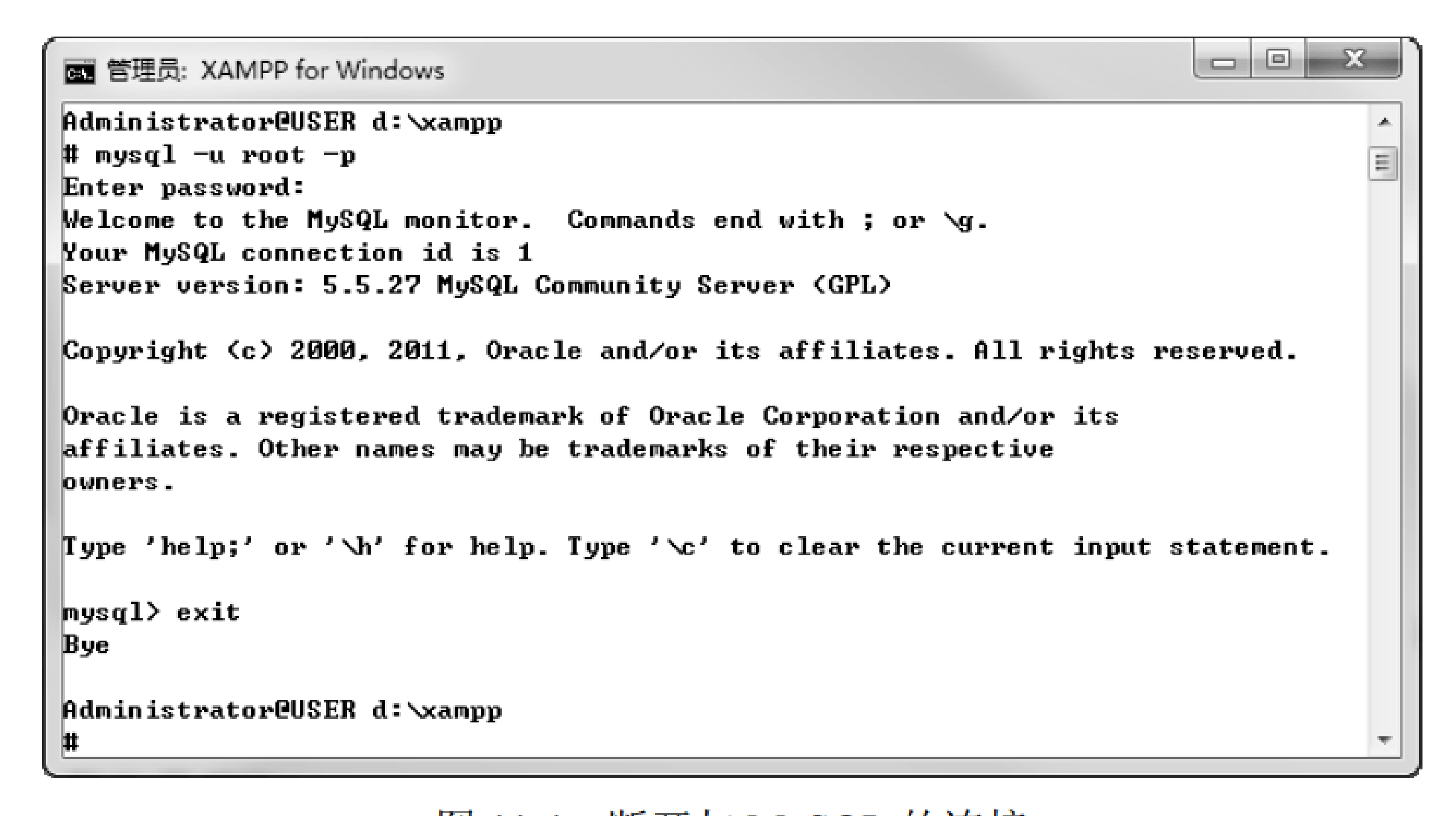


图 11.4　断开与 MySQL 的连接

以上介绍的就是连接与断开 MySQL 的操作。虽然这两步操作是非常简单的，但它们是非常重要的两个步骤，在 PHP 操作数据库管理系统的过程中同样是不可或缺的。

11.1.3　数据库操作

数据库管理系统通常使用 SQL（结构化查询语言）语句来操作。SQL 是大多数关系数据库管理系统所支持的工业标准。在接下来的大部分内容中，我们都将介绍 SQL 的使用。

1. 查看数据库

在一个数据库系统中可以有多个数据库，查看该数据库管理系统中存在的数据库可以使用如下命令：

```
SHOW DATABASES
```

SQL 命令中的关键字通常建议使用大写。一条 SQL 命令应该是以分号结束的。我们使用上面的命令来查看当前的 MySQL 中所有的数据库，执行的结果如下：

```
mysql> SHOW DATABASES;
+----------------------+
| Database             |
+----------------------+
| information_schema   |
| cdcol                |
| mysql                |
| performance_schema   |
| phpmyadmin           |
| webauth              |
+----------------------+
6 rows in set (0.00 sec)
```

从执行结果可以看到，现在的 MySQL 中存在 6 个数据库。

2. 创建数据库

为了使对数据库的操作不会影响到其他数据库而导致系统不稳定，通常会创建一个用于学习的数据库。创建数据库使用的命令如下：

```
CREATE DATABASE 数据库名称
```

如果期望创建的数据库名称已经存在，则会报告一条错误，如创建成功则报告影响到的数据行数。下面我们就来创建一个名为 mydatabase 的数据库，执行的结果如下：

```
mysql> CREATE DATABASE mydatabase;
Query OK, 1 row affected (0.00 sec)
```

我们可以再次查看数据库，运行结果如下：

```
mysql> SHOW DATABASES;
+----------------------+
| Database             |
+----------------------+
| information_schema   |
| cdcol                |
| mydatabase           |
```

```
| mysql                |
| performance_schema   |
| phpmyadmin           |
| webauth              |
+----------------------+
7 rows in set (0.00 sec)
```

从查询结果中可以看到，mydatabase 数据库已经被建立。

3. 删除数据库

与创建数据库对应的是删除数据库。当一个数据库不再需要使用的时候就可以将其删除。删除一个数据库的命令如下：

```
DROP DATABASE 数据库名称
```

例如删除我们新创建的 mydatabase 数据库的语句如下：

```
DROP DATABASE mydatabase;
```

该语句正确执行后 mydatabase 数据库会被删除，这里不做详细演示。

4. 取消操作

当我们在输入 SQL 命令的过程中出现错误时，可以通过在结尾加入“\c”来取消当前 SQL 命令。例如下面的语句：

```
DROP DATABASE mydatabase\c
```

11.1.4　数据表操作

在一个数据库管理系统中，通常会有多个数据库。直接用来存储数据的是数据库中的数据表，数据表就类似于一个二维表格。

1. 选择数据库

在前面的讲解过程中我们可以看到一个数据库管理系统中可以有多个数据库。要指定操作的数据库需要使用的命令如下：

```
USE 数据库名称;
```

这里我们可以选择一个存在的数据库。选择数据库管理系统中名为 mysql 的数据库需要执行的命令及执行的效果如下：

```
mysql> USE mysql;
Database changed
```

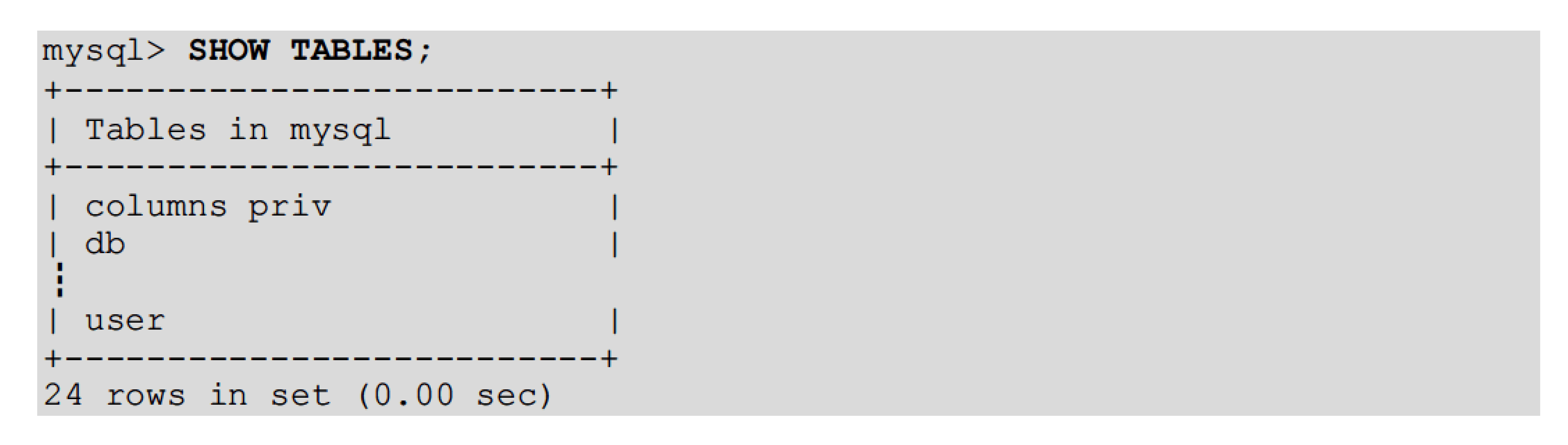
在成功选择数据库后客户端会返回 Database changed 的提示信息。

2. 显示数据库中的数据表

在成功选择一个数据库以后，我们就可以通过使用如下命令来查看该数据库中的数据表：

```
SHOW TABLES;
```

在执行该命令后通常会出现如下类似的信息：



```
mysql> SHOW TABLES;
+--------------------------+
| Tables in mysql          |
+--------------------------+
| columns priv             |
| db                       |
⋮
| user                     |
+--------------------------+
24 rows in set (0.00 sec)
```

限于篇幅问题，我们这里只列出了查询结果中的一部分。

3. 创建数据表和查看数据表结构

创建数据表可以使用以下语句：

```
CREATE TABLE 数据表名 (数据表定义);
```

数据表名可以使用不包括“\”、“/”和“.”的字符。数据表定义通常会定义该数据表的字段名、字段类型、数据宽度、是否可以为空、主键、自动增加和默认值。通常形式如下：

```
字段名 字段类型[(字段宽度)] [NOT NULL] [PRIMARY KEY] [AUTO_INCREMENT] [DEFAULT
默认值]
```

以上的形式可以在一条 SQL 语句中存在多次以定义数据表的多个字段。数据库管理系统中常用的数据类型及其取值返回如表 11.1 所示。

表 11.1　常用数据类型

类　　型	占用存储空间	说　　明
TINYINT	1	数值类型，有符号时的取值范围为－128～127，无符号时的取值范围为 0～255
SMALLINT	2	数值类型，有符号时的取值范围为－32768～32767，无符号时的取值范围为 0～65535
MEDIUMINT	3	有符号时的取值范围为－8388608～8388607，无符号时的取值范围为 0～16777215
INT	4	有符号时的取值范围为－2147483648～2147483647，无符号时的取值范围为 0～4294967295
BIGINT	8	有符号时的取值范围为－9223372036854775808～9223372036854775807，无符号时的取值范围为 0～18446744073709551615
DATE	3	日期和时间类型，支持的范围为“1000-01-01”到“9999-12-31”
TIME	3	日期和时间类型，支持的范围为“1000-01-01 00:00:00”到“9999-12-31 23:59:59”
CHAR(n)	n	字符类型，可以使用 n 限制允许的字符长度，取值范围为 0～255
VARCHAR(n)	n+1	字符类型，可以使用 n 限制允许的字符长度，取值范围为 0～65535

在了解了数据类型之后，我们就可以来创建一个最简单的数据表。为了数据的安全，我们选择在前面创建的 mydatabase 数据库中创建数据表，创建的代码如下：

```
CREATE TABLE simple_table (id INT);
```

以上代码执行后会输出以下信息：

```
mysql> CREATE TABLE simple_table(id INT);
Query OK, 0 rows affected (0.13 sec)
```

提示信息显示查询成功，我们可以通过如下命令来查看数据表的结构：

```
DESCRIBE 数据表名称;
```

查看 simple_table 数据表结构的语句及执行结果如下：

```
mysql> DESCRIBE simple_table;
+--------+---------+-------+-------+----------+-------+
| Field  | Type    | Null  | Key   | Default  | Extra |
+--------+---------+-------+-------+----------+-------+
| id     | int(11) | YES   |       | NULL     |       |
+--------+---------+-------+-------+----------+-------+
1 row in set (0.01 sec)
```

从执行结果可以看到该表的信息，由于我们只设定了列名称和类型，因此其他选项均使用默认值。其他选项被设置后的效果说明如下。

- NOT NULL：该选项被设置后该字段不能为空，例如可以限制用户名字段。
- PRIMARY KEY：该选择用来设置字段为主键，一个表中只可有一个主键，主键的值不可以重复。可以将 ID 类的字段设置为主键。
- DEFAULT：该选项用来设置字段的默认值。

现在我们就可以来完整地创建一个数据表。数据表的名称我们定为 complete_table，数据表的信息如表 11.2 所示。

表 11.2　complete_table 数据表信息

列名称	数据类型	可以为空	自动增加	默认值	主键
id	INT	NOT NULL	AUTO_INCREMENT	-	PRIMARY KEY
name	VARCHAR(20)	NOT NULL	-	-	-
sex	CHAR(4)	-	-	boy	-
register_time	DATE	-	-	-	-

根据表 11.2 我们就可以使用如下的 SQL 语句来创建 complete_table 数据表：

```
CREATE TABLE complete_table (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(20) NOT NULL,
    sex CHAR(4) DEFAULT ‘boy’,
    register_time DATE
);
```

以上语句的执行效果如下：

```
mysql> CREATE TABLE complete_table(
    -> id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    -> name VARCHAR(20) NOT NULL,
    -> sex CHAR(4) DEFAULT 'boy',
    -> register_time DATE
    -> );
```

```
Query OK, 0 rows affected (0.04 sec)
```

提示信息表示查询成功，说明 complete_table 数据表已经创建成功。查看 complete_table 数据表的信息如下：

```
mysql> DESCRIBE complete_table;
+-------------- +------------+------+------+---------+----------------+
| Field         | Type       | Null | Key  | Default | Extra          |
+-------------- +------------+------+------+---------+----------------+
| id            | int(11)    | NO   | PRI  | NULL    | auto_increment |
| name          | varchar(20)| NO   |      | NULL    |                |
| sex           | char(4)    | YES  |      | boy     |                |
| register time | date       | YES  |      | NULL    |                |
+-------------- +------------+------+------+---------+----------------+
4 rows in set (0.01 sec)
```

从查询结果可以清楚地看出，创建的 complete_table 数据表与我们在表 11.2 中所规定的信息是完全一致的。

4. 修改数据表结构

数据表在创建好之后通常不会频繁改动，但是在有些时候改动表结构是必要的。例如一个社交网站，记录用户的数据表在初期 id 的数据类型被设置为 TINYINT 型，如果注册的用户非常多，那么就有必要使用取值范围更大的数据类型。

修改数据表的结构可以使用以下形式的语句：

```
ALTER TABLE 数据表名 修改规则;
```

常用的修改规则有增加、修改和删除列，它们的形式如下：

```
ADD 列信息;
MODIFY 列名称 修改规则;
DROP 列名称;
```

例如为 complete_table 数据表添加 age 列，可以使用以下语句：

```
ALTER TABLE complete ADD age CHAR AFTER name;
```

以上语句执行后，会在 name 列后增加名为 age 数据类型为 char 的列，执行效果如下：

```
mysql> ALTER TABLE complete_table ADD age CHAR AFTER name;
Query OK, 0 rows affected (0.09 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

查看修改后的表结构如下：

```
mysql> DESCRIBE complete_table;
+---------------+------------+------+------+---------+----------------+
| Field         | Type       | Null | Key  | Default | Extra          |
+---------------+------------+------+------+---------+----------------+
| id            | int(11)    | NO   | PRI  | NULL    | auto increment |
| name          | varchar(20)| NO   |      | NULL    |                |
| age           | char(1)    | YES  |      | NULL    |                |
| sex           | char(4)    | YES  |      | boy     |                |
| register time| date        | YES  |      | NULL    |                |
+---------------+------------+------+------+---------+----------------+
5 rows in set (0.01 sec)
```

从查询结果可以看出在 name 列后加入了 age 列，如果新添加的列要作为第一列，则可以使用 FIRST 选项。接着我们可以 MODIFY 选项来将 age 列的数据类型修改为 TINYINT 并设置为不可为空，使用的语句及执行效果如下：

```
mysql> ALTER TABLE complete_table MODIFY age TINYINT NOT NULL;
Query OK, 0 rows affected (0.09 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

再次查看数据表结构如下：

```
mysql> DESCRIBE complete_table;
+---------------+-----------+------+------+---------+----------------+
| Field         | Type      | Null | Key  | Default | Extra          |
+---------------+-----------+------+------+---------+----------------+
| id            | int(11)   | NO   | PRI  | NULL    | auto increment |
| name          | varchar(20)| NO  |      | NULL    |                |
| age           | tinyint(4) | NO  |      | NULL    |                |
| sex           | char(4)    | YES |      | boy     |                |
| register time | date       | YES |      | NULL    |                |
+---------------+------------+------+------+---------+----------------+
5 rows in set (0.01 sec)
```

从修改后的表结构中可以看出，修改的结果与我们期望的结果是相同的。删除列的语句非常简单，删除 age 列可以使用如下的语句：

```
ALTER TABLE complete DROP age;
```

为了保持数据表的完整性，这里我们就不做实际演示。

5. 为表添加、修改和删除数据

在创建了期望的数据表之后，我们就可以向该数据表中添加数据了。向数据表中添加数据可以使用以下语句：

```
INSERT 数据表名 (列名,…) VALUES (值,…);
```

向 complete_table 中插入一行数据的语句如下：

```
INSERT complete_table (name,age,sex,register_time)
VALUES('Tom',15,'boy',CURDATE());
```

注意：CURDATE()为 MySQL 中的函数，它可以获取系统当前时间。

以上语句的执行效果如下：

```
mysql> INSERT complete_table (name,age,sex,register_time) VALUES(
    -> 'Tom',15,'boy',CURDATE());
Query OK, 1 row affected (0.01 sec)
```

同创建数据表类似，可以通过逗号来分隔每行数据的值，以使用一条 SQL 语句向数据表中插入多行数据：

```
mysql> INSERT complete_table(name,age,sex,register_time)VALUES
    -> ('Jim','16','boy',CURDATE()),
    -> ('Mary','13','girl',CURDATE()),
    -> ('Anne','14','girl',CURDATE());
Query OK, 3 rows affected (0.04 sec)
```

```
Records: 3  Duplicates: 0  Warnings: 0
```

到此为止，我们为 complete_table 数据表中加入了 4 行数据。有些数据会经常改变，我们就可以通过使用以下语句来更改数据：

```
UPDATE 数据表名 SET 列名称=值 [WHERE 设置条件];
```

将数据表中 Jim 的年龄由 16 改为 15 的语句如下：

```
UPDATE complete_table SET age=15 WHERE name='Jim';
```

注意：如果不指定设置条件，则会更改数据表中所有的记录。

以上语句的执行效果如下：

```
mysql> UPDATE complete_table SET age=15 WHERE name='Jim';
Query OK, 1 row affected (0.04 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

删除数据表中的数据可以使用以下的形式：

```
DELETE FROM 数据表名 [WHERE 删除条件];
```

删除数据表中 Mary 的记录可以使用以下的语句：

```
DELETE FROM complete_table WHERE name='Mary';
```

注意：如果不指定删除条件，则会删除数据表中所有的记录。

以上语句的执行效果如下：

```
mysql> DELETE FROM complete_table WHERE name='Mary';
Query OK, 1 row affected (0.03 sec)
```

对数据表中记录的操作这里只做了简单介绍，如果读者想要深入学习，可以参考相关专业的书籍。

6. 删除数据表

当一个数据表不再需要的时候就可以将其删除，可以使用以下形式：

```
DROP TABLE 数据表名;
```

注意：该语句会删除表中的数据及表的结构，需要谨慎使用。

删除 complete_table 数据表可以使用以下语句：

```
DROP TABLE complete_table;
```

以上语句的执行效果如下：

```
mysql> DROP TABLE complete_table;
Query OK, 0 rows affected (0.03 sec)
```

执行成功后 complete 数据表即被删除，查看 mydatabase 数据库中的数据表：

```
mysql> SHOW TABLES;
```

```
+----------------------+
| Tables_in_mydatabase |
+----------------------+
| simple_table         |
+----------------------+
1 row in set (0.00 sec)
```

从查询结果可以看出，mydatabase 数据库中只存在 simple_table 数据表。

11.1.5　查询数据操作

查询数据库操作是数据库管理系统中使用最频繁的操作。本节就简单介绍一下查询数据的操作。在进行查询操作之前，我们应该建立一个如表 11.3 所示的简单数据表。

表 11.3　report_card 表

student_id	name	chinese	math	english	total_points
1	Jim	85	63	92	240
2	Tom	66	75	89	230
3	Anne	92	76	67	235
4	Mary	78	90	73	241
5	Allen	90	77	76	243

查询整个表中的数据可以使用以下语句：

```
SELECT * FROM 数据表名;
```

查看 report_card 数据表信息的语句及执行效果如下：

```
mysql> SELECT * FROM report_card;
+-----------+------+-----------+-------+----------+--------------+
| student_id| name | chinese   | math  | english  | total_points |
+-----------+------+-----------+-------+----------+--------------+
|         1 | Jim  |     85    |  63   |     92   |      240     |
|         2 | Tom  |     66    |  75   |     89   |      230     |
|         3 | Anne |     92    |  76   |     67   |      235     |
|         4 | Mary |     78    |  90   |     73   |      241     |
|         5 | Allen |    90    |  77   |     76   |      243     |
+-----------+------+-----------+-------+----------+--------------+
5 rows in set (0.00 sec)
```

查看数据表中指定的列可以使用以下形式：

```
SELECT 列名称[,…] FROM 数据表名;
```

查看 report_card 数据表中的 name 和 total_points 列的语句及执行效果如下：

```
mysql> SELECT name,total_points FROM report_card;
+-------+--------------+
| name  | total_points |
+-------+--------------+
| Jim   |     240      |
| Tom   |     230      |
| Anne  |     235      |
| Mary  |     241      |
| Allen |     243      |
```

```
+--------+--------------+
5 rows in set (0.00 sec)
```

与修改数据表类似，查询数据也可以通过使用 WHERE 设置查询条件，形式如下：

```
SELECT {列名称[,…]|*} FROM 数据表名 WHERE 查询条件;
```

查询 report_card 数据表中 Anne 成绩使用的语句及执行效果如下：

```
mysql> SELECT * FROM report_card WHERE name='Anne';
+-------------+------+---------+------+-----------+-------------+
| student_id  | name | chinese | math | english   | total_points |
+-------------+------+---------+------+-----------+-------------+
|          3  | Anne |      92 |   76 |    67     |       235 |
+-------------+------+---------+------+-----------+-------------+
1 row in set (0.00 sec)
```

查询 chinese 列值大于 80 的数据并显示其 name 列的语句以及执行效果如下：

```
mysql> SELECT name FROM report_card WHERE chinese>80;
+----------+
| name     |
+----------+
| Jim      |
| Anne     |
| Allen    |
+----------+
3 rows in set (0.00 sec)
```

查询数据操作还可以使用 ORDER BY 选项对查询结果排序，语句形式如下：

```
SELECT {列名称[,…]|*} FROM 数据表名 ORDER BY 列名称[,…] [ASC|DESC[,…]];
```

ASC 和 DESC 选项用于控制排序方式，默认为按照升序排列（ASC）。将表 report_card 按照 total_points 列降序排列使用的语句以及运行效果如下：

```
mysql> SELECT * FROM report_card ORDER BY total_points DESC;
+-------------+--------+-----------+------+--------+-------------+
| student_id  | name   | chinese   | math | english| total_points |
+-------------+--------+-----------+------+--------+-------------+
|         5   | Allen  |      90   |  77  |   76   |      243     |
|         4   | Mary   |      78   |  90  |   73   |      241     |
|         1   | Jim    |      85   |  63  |   92   |      240     |
|         3   | Anne   |      92   |  76  |   67   |      235     |
|         2   | Tom    |      66   |  75  |   89   |      230     |
+-------------+--------+-----------+------+--------+-------------+
5 rows in set (0.00 sec)
```

通过使用 LIMIT 选项可以设置输出查询结果的条数，语法形式如下：

```
SELECT {列名称[,…]|*} FROM 数据表名 LIMIT 行数;
```

将上面排序的结果只显示前 3 行的语句及执行效果：

```
mysql> SELECT * FROM report_card ORDER BY total_points DESC LIMIT 3;
+-------------+--------+-----------+------+--------+-------------+
| student_id  | name   | chinese   | math | english| total_points |
+-------------+--------+-----------+------+--------+-------------+
|         5   | Allen  |      90   |  77  |   76   |      243     |
|         4   | Mary   |      78   |  90  |   73   |      241     |
```

```
|         1    | Jim      |      85     |   63  |    92    |       240     |
+-------------+--------+-----------+------+--------+--------------+
3 rows in set (0.00 sec)
```

以上介绍的就是查询数据库操作常用的一些操作，如果读者想要学习更多的数据库查询操作的知识，可以参考相关专业书籍。

11.1.6　使用 phpMyAdmin 管理数据库

在前面的学习过程中，我们一直都是在命令行中进行一系列的操作。以命令行的方式操作数据库虽然效率很高，但是对于初学者或者业余爱好者来说都是不太友好的。phpMyAdmin 是使用 PHP 编写的一套用户界面友好的数据库管理工具。在我们使用的 XAMPP 集成环境中就包含这个工具。因为类似的工具都对数据库的操作进行了简化，虽然操作都非常简单，但是对于我们学习 PHP 的读者来说并不是最合适的。因此本章才会以命令行方式为主，而不是使用 phpMyAdmin。下面就来简单介绍一下 phpMyAdmin。

1. 启动 phpMyAdmin

启动 phpMyAdmin 的方式有两种，一种为直接使用浏览器访问地址 http://localhost/phpmyadmin/即可。第二种方式为从 XAMPP 控制面板启动，操作如图 11.5 所示。

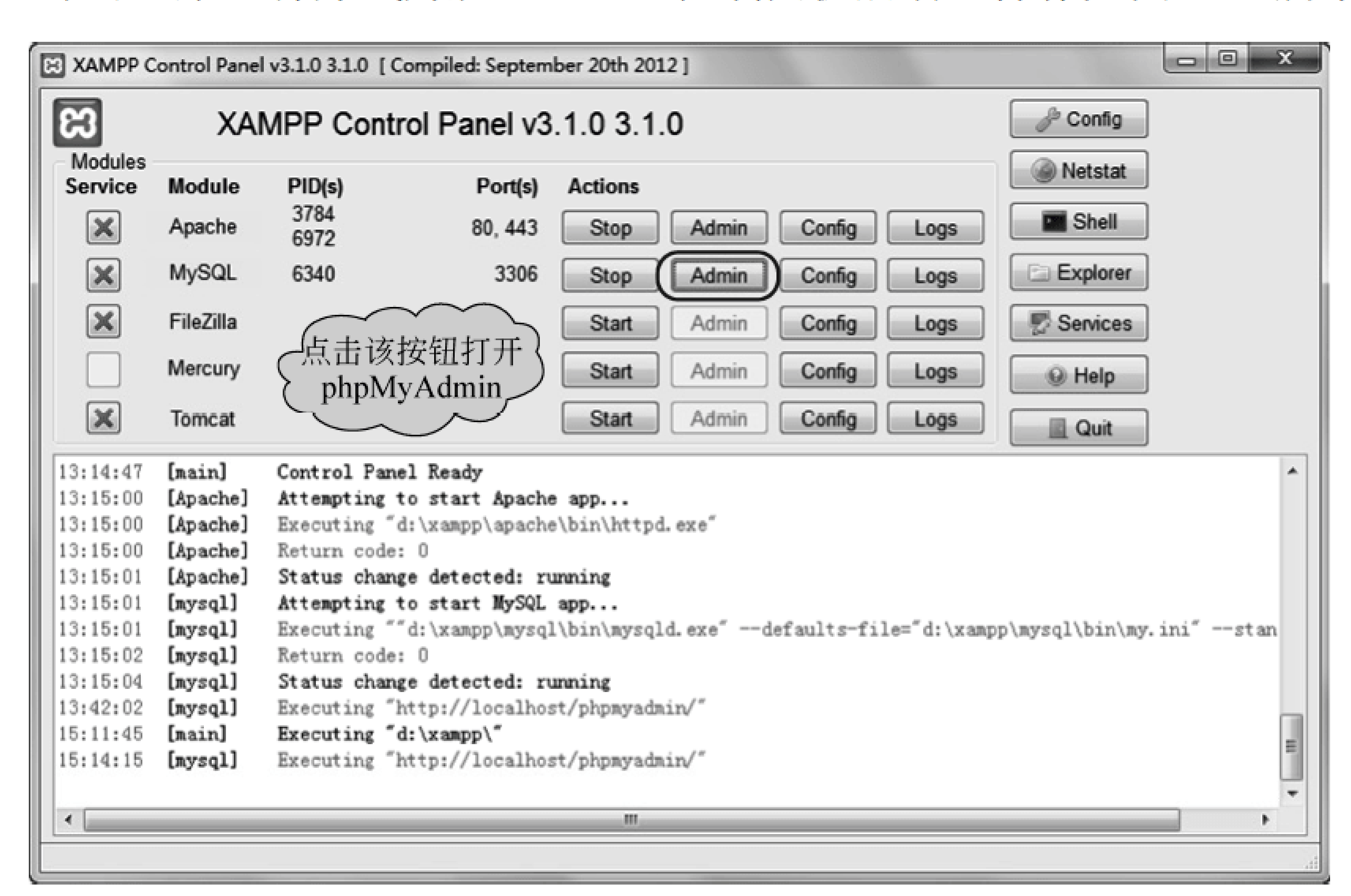


图 11.5　使用 XAMPP 控制面板打开 phpMyAdmin

phpMyAdmin 的主界面如图 11.6 所示。

phpMyAdmin 主界面的左侧列出的是当前系统中的数据库。大部分右侧则是一些容易理解的设置和相关的信息。

2. 管理数据库

在 phpMyAdmin 主界面左侧的数据库中单击一个数据库即可查看该数据库，这里以查看 mydatabase 数据库为例，该数据库的信息如图 11.7 所示。

单击其中的数据表名称即可查看该数据表中的数据。这里以查看 report_card 数据表为例，如图 11.8 所示。



图 11.6　phpMyAdmin 主界面



图 11.7　mydatabase 数据库的信息

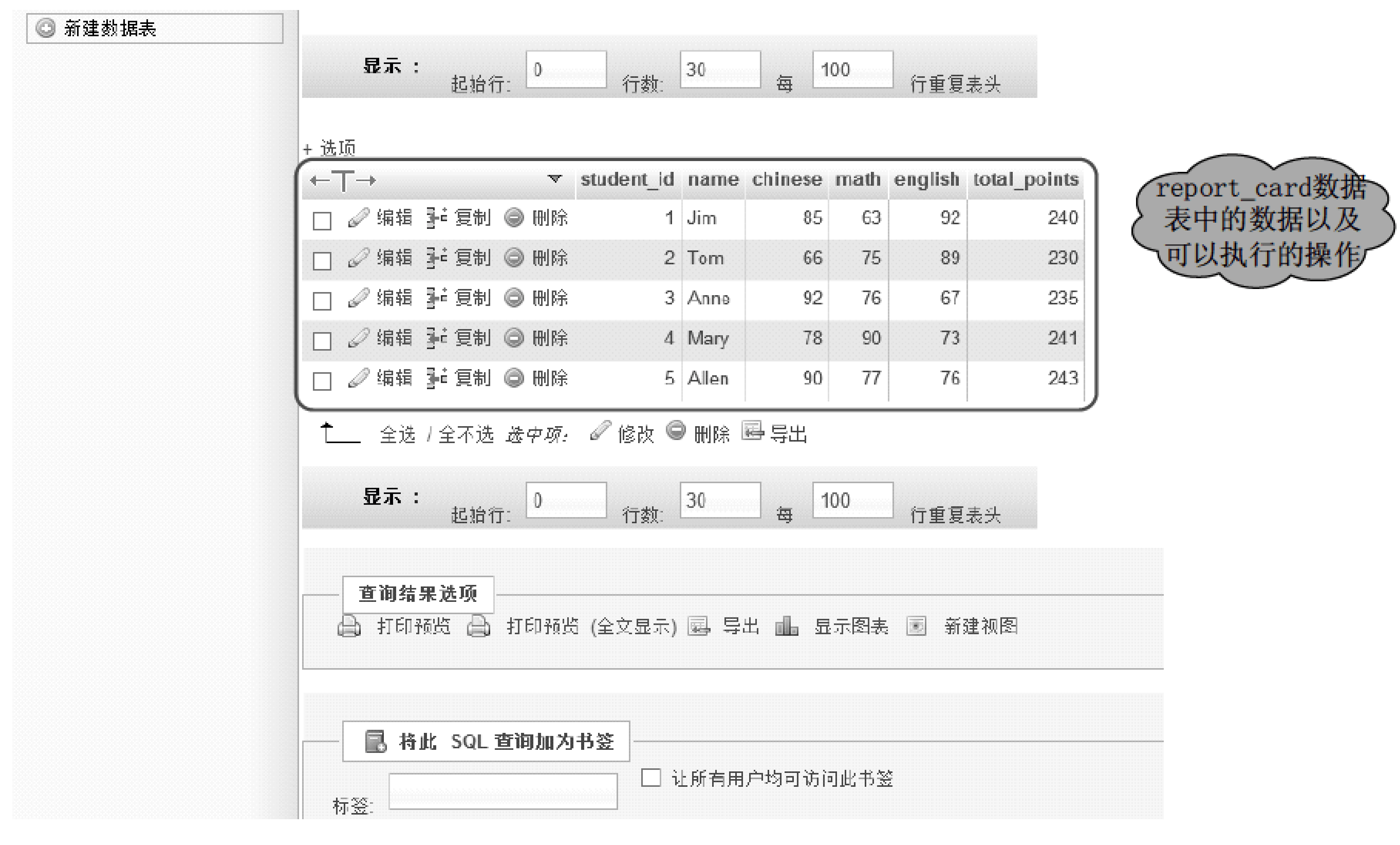


图 11.8　report_card 数据表

单击数据表对应操作中的“结构”可以查看数据表的结构，这里以 complete_table 为

例，如图 11.9 所示。



图 11.9　complete_table 数据表的结构

单击数据表对应操作中的“插入”按钮，可以为数据表添加数据，这里以 report_card 为例，如图 11.10 所示。

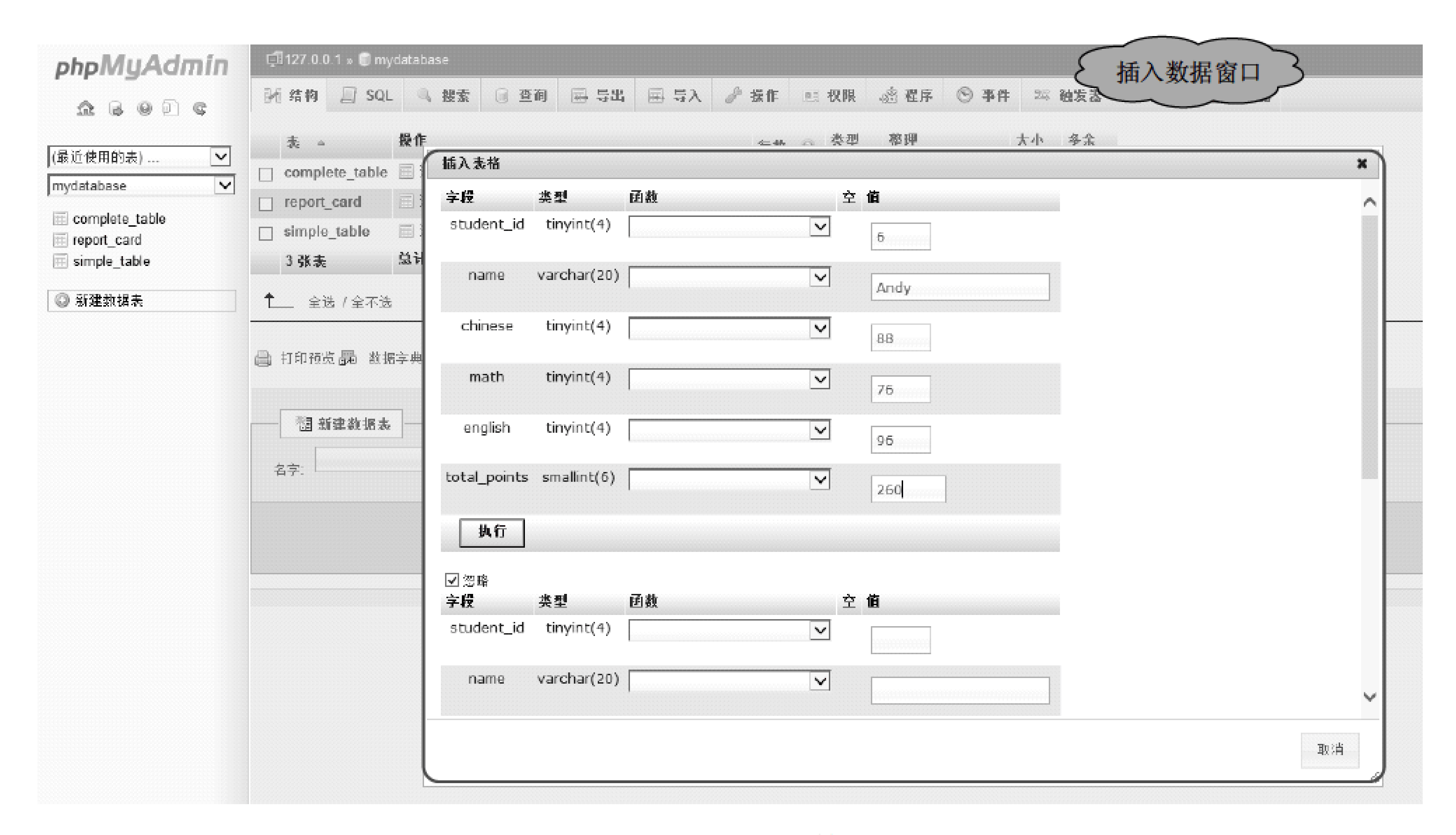


图 11.10　插入数据

从以上一系列的介绍可以看出，phpMyAdmin 的使用是非常简单和直观的，这里不多做介绍，有兴趣的读者可以自己尝试其他操作。

11.2　使用 PHP 操作数据库

在 11.1 节中介绍了使用命令行方式或者 phpMyAdmin 工具来管理数据库。本节我们来介绍使用 PHP 来实现操作数据库。

11.2.1　PHP 操作数据库流程

标准的数据库操作流程分为以下 3 步：

（1）打开与数据库的连接。

（2）对数据库进行相关操作。

（3）关闭与数据库的连接。

打开与数据库的连接可以使用 mysql_connect()函数，该函数原型如下：

```
resource mysql_connect ([ string $server [, string $username [, string
$password [, bool $new_link [, int $client_flags ]]]]] )
```

参数 server 为 MySQL 服务器名，默认为“localhost:3306”；参数 username 为数据库的用户名；参数 password 为数据库对应用户名的密码；参数 new_link 用来控制是否打开新的连接；参数 client_flags 用于传送客户端标记，可以为以下参数。

- MYSQL_CLIENT_COMPRESS：使用压缩的通信协议。
- MYSQL_CLIENT_IGNORE_SPACE：允许在函数名后有空格。
- MYSQL_CLIENT_INTERACTIVE：允许断开连接之前等候 interactive_timeout 时间。
- MYSQL_CLIENT_SSL：使用 SSL 加密。

mysql_connect()函数通常只使用前三个参数。

关闭数据库连接可以使用 mysql_close()函数，它的原型如下：

```
bool mysql_close ([ resource $link_identifier ] )
```

参数 link_identifier 为打开的数据库连接，如果省略该参数则关闭上一个打开的数据库连接。该函数不是必须要调用的，当程序执行完毕后，数据库管理系统会自动关闭非持久性连接。

【示例 11-1】以下代码演示使用 mysql_connect()函数连接 MySQL 服务器。

```
<?php
    $link=mysql_connect('localhost:3306','root','')or
            die('数据库连接失败！');     //连接失败则终止执行
    echo '数据库连接成功！';             //连接成功则输出成功信息
    mysql_close($link);                  //关闭数据库连接
?>
```

以上代码运行结果如图 11.11 所示。

服务器名称、用户名和密码错误均会导致连接失败。但是所有参数为空的时候有可能会连接成功，因为 MySQL 可能会有一个任意用户名并且没有密码的用户，但是这些用户可能会没有足够的权限去操作 MySQL。



图 11.11　运行结果

11.2.2　查询数据库

在成功连接数据库之后即可对数据库进行查询操作，这就类似于在命令行方式下成功登录后向数据库发送 SQL 语句。

1. 发送 SQL 查询

在 PHP 中向数据库发送 SQL 查询可以使用 mysql_query()函数，该函数会向数据库发送 SQL 查询并将查询的结果返回。mysql_query()函数的原型如下：

```
resource mysql_query ( string $query [, resource $link_identifier ] )
```

参数 query 即为 SQL 语句；可选参数 link_identifier 为打开的数据库连接，如果省略则

使用上一个打开的连接，没有打开的连接，则该函数会无参数调用 mysql_connect()函数。以下代码发送 SQL 查询来查看 MySQL 中存在的数据库：

```
mysql_query('SHOW DATABSES',$link)
```

查询的结果会以资源类型返回，我们需要使用专门的函数来读取返回的结果。

2. 获取结果集

mysql_fetch_row()、mysql_fetch_assoc()和 mysql_fetch_array()函数可以从返回的结果集中取出一行作为数组元素。mysql_fetch_row()函数会返回索引数组；mysql_fetch_assoc()函数会返回关联数组；mysql_fetch_array()函数可能返回关联数组或者索引数组。以上 3 个函数的原型如下：

```
array mysql_fetch_row ( resource $result )
array mysql_fetch_assoc ( resource $result )
array mysql_fetch_array ( resource $result [, int $ result_type ] )
```

3 个函数中的参数 result 为 SQL 查询的结果集。mysql_fetch_array()函数的参数 result_type 可以为以下取值。

- MYSQL_ASSOC：只返回关联数组。
- MYSQL_BOTH：该选项为默认值，返回有数字和字符索引的关联数组。
- MYSQL_NUM：只返回索引数组。

【示例 11-2】以下代码演示使用 mysql_fetch_row()函数取得 SQL 查询的结果集。

```
<?php
    $link=mysql_connect('127.0.0.1','root','')or die('数据库连接失败！');
    $res=mysql_query('SHOW DATABASES',$link);       //发送 SQL 查询
    while($dbs=mysql_fetch_row($res))              //遍历查询结果集
        echo $dbs[0].'<br />';
?>
```

注意：同字符串操作类似，mysql_connect 返回的资源中也有一个类似的指针。从结果集获取数据的函数通常会移动指针。因此上面的示例中可以使用 while 循环来遍历结果集。

代码运行结果如图 11.12 所示。

运行结果输出的即为 MySQL 中存在的数据库。

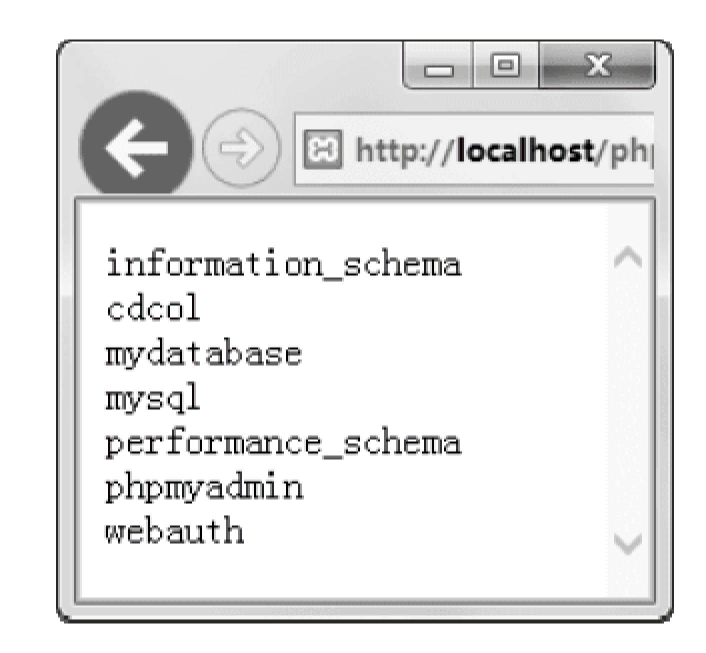


图 11.12　运行结果

3. 选择数据库

在成功连接数据库管理系统后，通常会选择一个数据库进行操作。但是通常不会使用 mysql_query()函数发送 SQL 查询来选择一个数据库，而是使用专门的 mysql_select_db()函数来选择一个数据库，它的原型如下：

```
bool mysql_select_db ( string $database_name [, resource $
link_identifier  ] )
```

参数 database_name 即为需要选择的数据库；可选参数 link_identifier 为打开的数据库

连接；如果省略则使用上一个打开的连接，没有打开的连接，则该函数会无参数调用 mysql_connect()函数。在该函数执行成功后，后面的 mysql_query()函数均对已选择的数据库进行查询。

【示例 11-3】以下代码演示使用 mysql_select_db()函数选择一个数据库。

```
<?php
    $name='mydatabase';              //定义数据库名
    $link=mysql_connect('127.0.0.1','root','')or die('数据库连接失败！');
    if(mysql_select_db($name))       //选择数据库并判断是否成功
        echo "已成功选择{$name}。";
?>
```

代码运行结果如图 11.13 所示。

如果期望选择的数据库并不存在，mysql_select_db()函数会返回 FALSE。

4. 查询数据

在选择数据库后即可查询其中的数据。

【示例 11-4】以下代码演示列出 report_card 数据表中的数据。

```
<?php
    $query='SELECT * FROM report_card';
    $link=mysql_connect('127.0.0.1','root','')or die('数据库连接失败！');
    mysql_select_db('mydatabase')or die('选择数据库失败！');
    $res=mysql_query($query,$link);               //发送查询语句
    while($rows=mysql_fetch_row($res)){           //遍历结果集
        foreach($rows as $k=>$v)
            echo "{$k}=>{$v} ";
        echo '<br />';
    }
?>
```

代码运行结果如图 11.14 所示。



图 11.13　运行结果

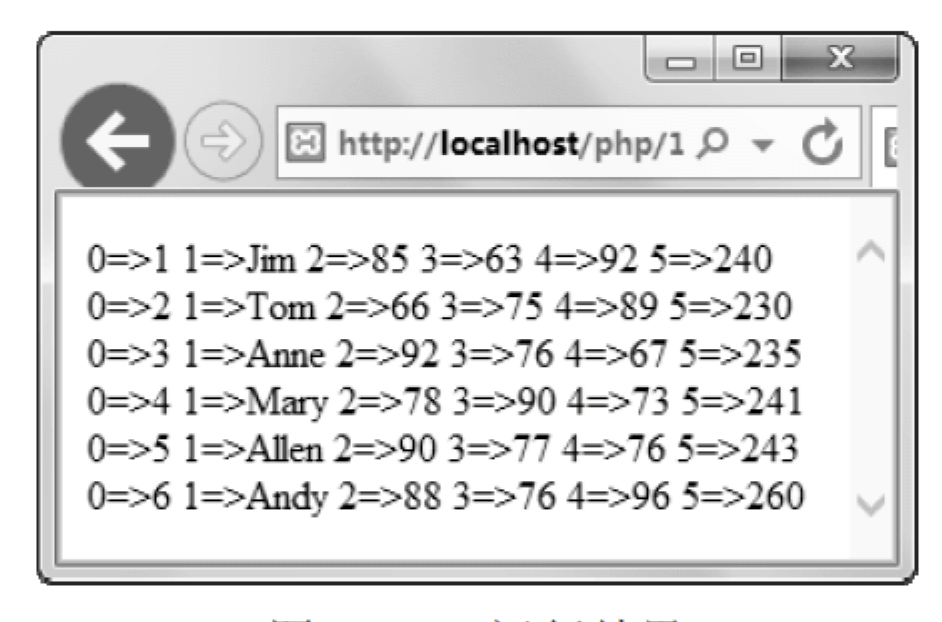


图 11.14　运行结果

从运行结果可以看到，虽然表中的数据被无误地输出了，但是并不容易阅读。下面我们就将这些数据装入表格。

【示例 11-5】以下代码演示将示例 11-4 代码中的查询结果装入表格。

```
<?php
    $query='SELECT * FROM report_card';
    $link=mysql_connect('127.0.0.1','root','')or die('数据库连接失败！');
    mysql_select_db('mydatabase')or die('选择数据库失败！');
```

```
    $res=mysql_query($query,$link);                 //发送 SQL 查询
    echo '<table border=1>';                        //输出 HTML 标签
    while($rows=mysql_fetch_row($res)){             //遍历结果集
        echo '<tr>';                                //输出 HTML 标签
        foreach($rows as $k=>$v)
            echo "<td>{$v}</td>";
        echo '</tr>';                               //输出 HTML 标签
    }
    echo '</table>';                                //输出 HTML 标签
?>
```

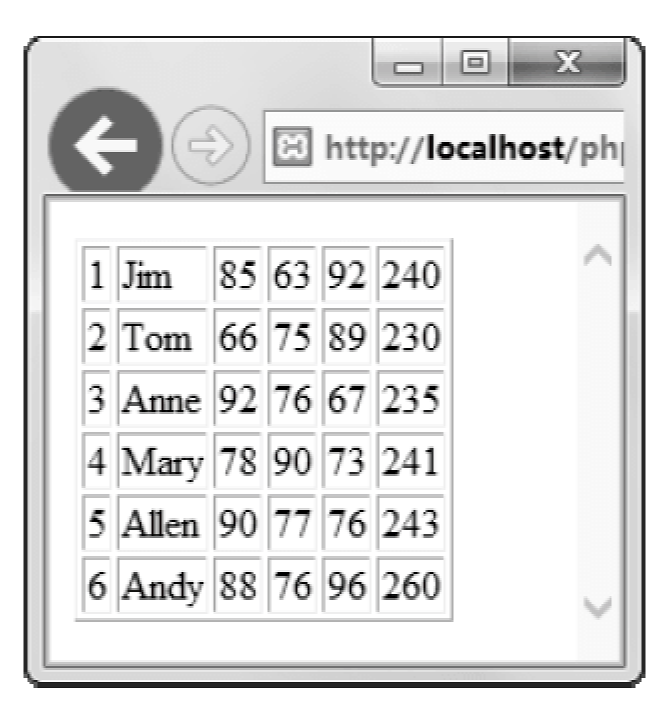
代码运行结果如图 11.15 所示。

从运行结果可以看到，将查询到的数据装入表格后更加直观了，但是这些数据所表示的含义并不直观，下面我们就来将列名称加入表格。

【示例 11-6】以下代码演示将列名称加入示例 11-5 所输出的表格中。

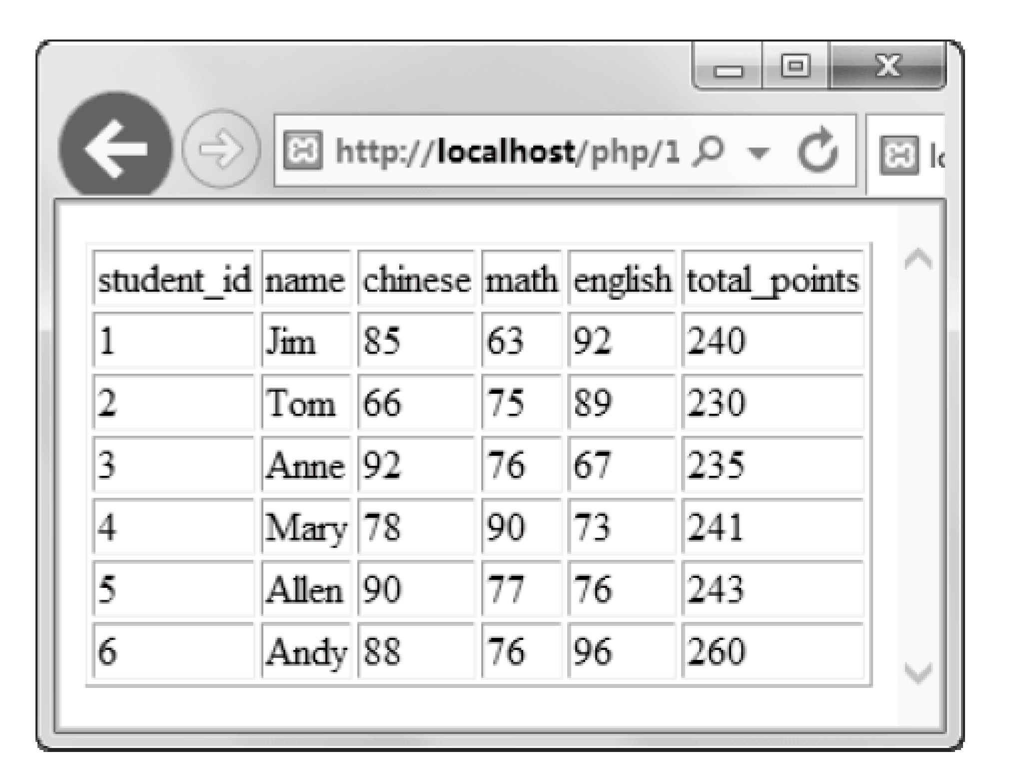
```
<?php
    $query='DESCRIBE report_card';              //查询表结构
    $link=mysql_connect('127.0.0.1','root','')or die('数据库连接失败！');
    mysql_select_db('mydatabase')or die('选择数据库失败！');
    $res=mysql_query($query,$link);
    echo '<table border=1><tr>';                //输出 HTML 标签
    while($rows=mysql_fetch_row($res)){
        echo "<td>{$rows[0]}</td>";             //输出列名称
    }
    echo '</tr>';
    $query='SELECT * FROM report_card';         //查询表数据
    $res=mysql_query($query,$link);             //发送 SQL 查询
    while($rows=mysql_fetch_row($res)){         //遍历结果集
        echo '<tr>';                            //输出 HTML 标签
        foreach($rows as $k=>$v)
            echo "<td>{$v}</td>";
        echo '</tr>';                           //输出 HTML 标签
    }
    echo '</table>';                            //输出 HTML 标签
?>
```

代码运行结果如图 11.16 所示。



1	Jim	85	63	92	240
2	Tom	66	75	89	230
3	Anne	92	76	67	235
4	Mary	78	90	73	241
5	Allen	90	77	76	243
6	Andy	88	76	96	260

图 11.15　运行结果



student_id	name	chinese	math	english	total_points
1	Jim	85	63	92	240
2	Tom	66	75	89	230
3	Anne	92	76	67	235
4	Mary	78	90	73	241
5	Allen	90	77	76	243
6	Andy	88	76	96	260

图 11.16　运行结果

从运行结果可以看到，该数据表中数据的显示已经比较完美了。

5. 获取查询影响的记录行数

在进行一些 SQL 查询的时候，可能会对数据表中的数据产生影响，例如修改和插入数据。本节我们将主要介绍 mysql_affected_rows()函数，它可以记录上一次查询操作影响的记录行数。该函数的原型如下：

```
int mysql_affected_rows ([ resource $link_identifier ] )
```

可选参数 link_identifier 为打开的数据库连接，省略该函数则会无参数调用 mysql_connect()函数。

【示例 11-7】以下代码演示使用 mysql_affected_rows()函数获取上一次查询操作影响的记录行数。

```
<?php
    $query='DELETE FROM report_card WHERE student_id>3';
    $link=mysql_connect('127.0.0.1','root','')or die('数据库连接失败！');
    mysql_select_db('mydatabase',$link)or die('选择数据库失败！');
    $res=mysql_query($query,$link);                  //SQL 查询
    echo '上一次查询操作影响了'.mysql_affected_rows($link).'行记录。';
                                                     //获取影响行数
?>
```

代码运行结果如图 11.17 所示。



图 11.17　运行结果

该示例中的查询语句删除了数据表中 student_id 大于 3 的数据。到此为止，查询数据库部分的操作就介绍完毕了。当然，还有一些其他操作没有做介绍，因为都与前面介绍过的示例类似，读者可以轻易掌握。

11.3　小　　结

本章主要介绍了 MySQL 数据库管理系统的一些常用操作及使用 PHP 对 MySQL 进行操作。虽然，在本章中使用大部分篇幅介绍 SQL 语句，但是 SQL 语句的内容远远不止这么多，读者可以根据自身情况加强 SQL 语句的学习。使用 PHP 操作 MySQL 部分则比较容易学习，只要可以熟练使用 SQL 语句则可以很轻松地掌握。

11.4　本 章 习 题

1. 使用命令行窗口连接数据库管理系统。
2. 在 MySQL 中创建一个名为 newdatabase 的新数据库。
3. 在新创建的 newdatabase 数据库中创建一个新数据表 newtable 并定义如表 11.4 所示

的表结构。

表 11.4　newtable 表结构

列名称	数据类型	可以为空	自动增加	主　键
id	INT	NOT NULL	AUTO_INCREMENT	PRIMARY KEY
name	VARCHAR(20)	NOT NULL	-	-
hobby	CHAR(20)	NOT NULL	-	-

4. 向新创建的 newtable 表中添加如表 11.5 所示的数据。

表 11.5　newtable 表数据

id	name	hobby	id	name	hobby
1	Jim	walk	4	Mary	shopping
2	Tom	basketball	5	Allen	computer game
3	Anne	cartoon			

5. 设计 PHP 程序将表 newtable 中的数据作为表格内容输出到浏览器。

第 12 章　Cookie 和 Session

Cookie 和 Session 是 Web 中非常重要的两个技术。因为目前大多数 Web 网站是通过 HTTP 协议传输的，而 HTTP 协议是无状态的，也就是不会记录之前的操作，每次 HTTP 请求都可以视为独立的。Cookie 和 Session 就是用来记录一些操作以供之后的请求使用。

12.1　Cookie 技术

本节将会介绍 Cookie 的实质，以使读者理解 Cookie 的运作原理，然后再介绍 Cookie 在 PHP 中如何被使用，以及如何应用在实际案例中。

12.1.1　什么是 Cookie

Cookie 实际上是存储在客户端的普通文件。可以通过查看 Cookie 所在的位置来找到这些文件，如图 12.1 所示。

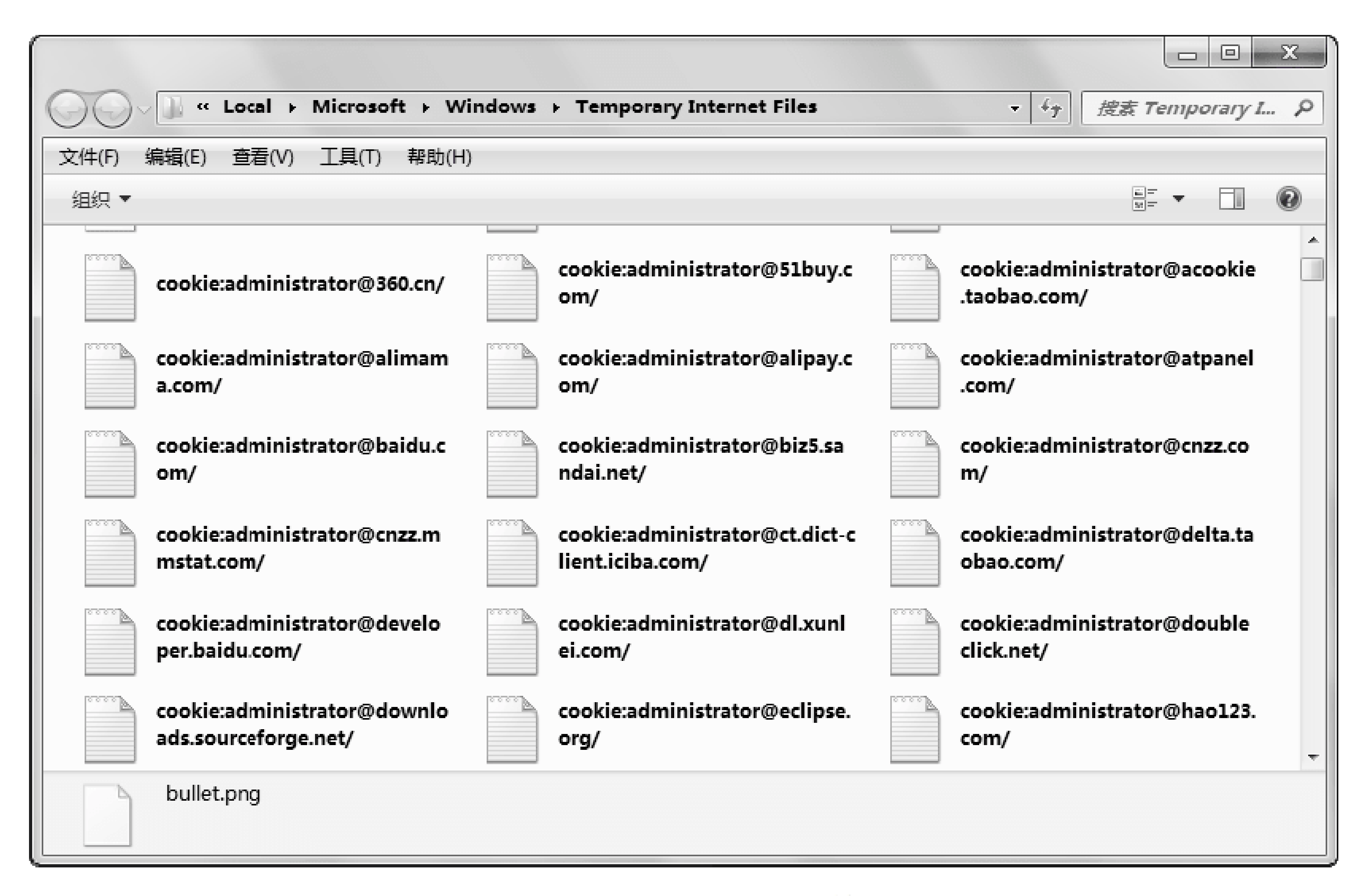


图 12.1　Cookie 文件

我们可以查看其中一个 Cookie 文件的属性及其中的内容，如图 12.2 所示。

从 Cookie 文件属性中也可以得到关于 Cookie 文件的一些信息。例如，Cookie 文件的类型、缓存名称和过期时间。过期时间是 Cookie 的一项重要属性，它用来规定该 Cookie 可以被读取的截止时间。Cookie 在未被浏览器禁止使用的情况下，当我们请求一个站点的时候，浏览器会检查磁盘中是否存在该站点的 Cookie，如果存在并且未过期，则从 Cookie 文件中读取相关信息并将这些信息发送到指定站点，站点则根据 cookie 信息返回特定的网

页。虽然 Cookie 文件是一个普通的文件，通常情况下并不容易阅读，我们也可以从图 12.2 中看出来。



图 12.2　Cookie 文件属性及内容

12.1.2　设置 Cookie

Cookie 在使用前需要生成 Cookie 文件，也就是设置 Cookie。在 PHP 中，可以使用 setcookie 来创建一个 Cookie，该函数的原型如下：

```
bool setcookie ( string $name [, string $value [, int $expire = 0 [, string
$path [, string $domain [, bool $secure = false [, bool $httponly =
false ]]]]]] )
```

参数 name 用来指定 Cookie 的名称；可选参数 value 用来指定 Cookie 中 name 对应的值；可选参数 expire 用来指定 Cookie 的过期时间，单位为秒；可选参数 path 用来设置 Cookie 可以使用的范围，默认为 Cookie 文件的当前目录；可选参数 domain 用来设置可以被使用的域名范围；可选参数 secure 用来规定是否只通过 HTTPS 协议传输；参数 httponly 用来规定是否只通过 HTTP 协议传输。

【示例 12-1】以下代码演示设置一个 Cookie。

```
<?php
    setcookie('mycookie','true',time()+60*60)or die(); //设置 Cookie
    echo 'Cookie 设置成功。';                          //输出提示信息
?>
```

代码运行结果如图 12.3 所示。代码中 time()函数用来获取当前时间，那么过期时间“time()+60*60”则是当前时间 1 小时后。运行结果提示 Cookie 设置成功，我们可以找到该 Cookie 文件并查看其属性，如图 12.4 所示。

如果不为 Cookie 文件设置过期时间，则 Cookie 的失效期为直到浏览器关闭。在代码中一次可以设置多个 Cookie，如果为同名的 Cookie 设置新值则为改写同名 Cookie 项的值。

12.1.3　读取 Cookie

在设置 Cookie 文件后则可以使用 Cookie，也就是读取 Cookie 文件中的信息。在 PHP



图 12.3　运行结果

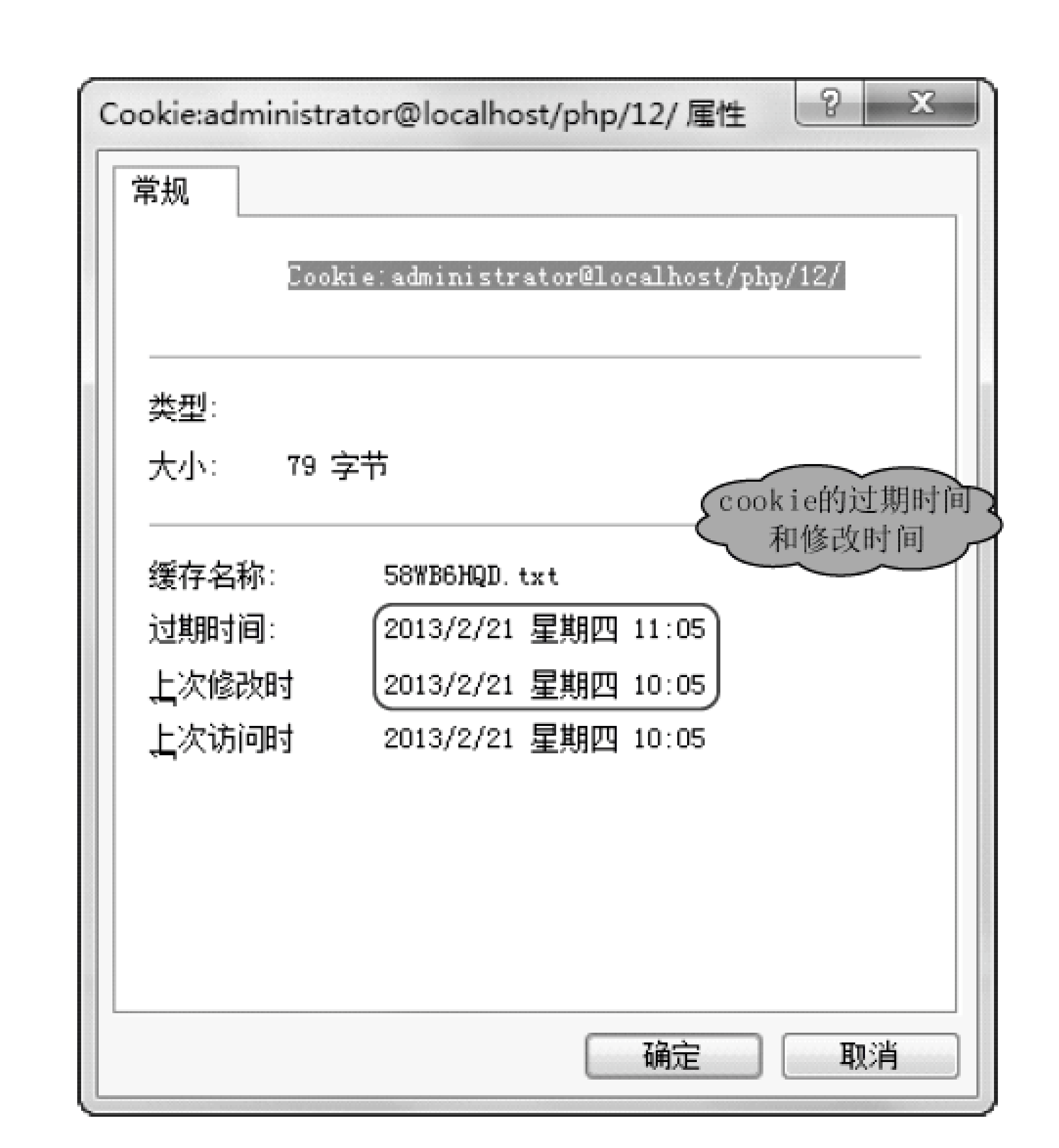


图 12.4　示例 12-1 程序执设置的 Cookie 文件属性

中，读取 Cookie 需要使用一个全局变量$_COOKIE。该变量是一个数组，所有设置的 Cookie 键值对均会加入该数组。

【示例 12-2】以下代码演示设置多个 Cookie 并输出全局变量$_COOKIE 的信息。

```
<?php
    setcookie('name','Tom',time()+60*60);          //设置 Cookie
    setcookie('register',time(),time()+60*60);     //设置 Cookie
    echo '全局变量$_COOKIE: ';
    print_r($_COOKIE);                             //输出全局变量$_COOKIE 的信息
?>
```

代码运行结果如图 12.5 所示。

从运行结果中我们并没有看到设置的 Cookie 信息，这是因为该程序执行前并没有该 Cookie，程序执行后会设置一个 Cookie。但我们刷新这个页面后，Cookie 的内容才会被读取，如图 12.6 所示。

从上面的演示可以看到，$_COOKIE 全局变量就是一个数组，那么我们就可以很容易地访问到其中的元素。

【示例 12-3】以下代码演示读取 Cookie 中的内容。

```
<?php
    echo '当前用户名为: '.$_COOKIE['name'];        //读取$_COOKIE 数组元素
?>
```

代码运行结果如图 12.7 所示。




图 12.5　运行结果



图 12.6　刷新页面后读取到的 Cookie



图 12.7　运行结果

由于本章示例中的源文件均处于同一路径下，因此该示例可以读取到示例 12-2 设置的

Cookie。

12.1.4　删除 Cookie

如 Cookie 不需要再使用未过期的 Cookie 时可以使 Cookie 文件立即过期，即删除 Cookie。删除 Cookie 有两种常用的方式，一种为 Cookie 项设置空值，另一种为使 Cookie 项过期。为 Cookie 项赋空值，可以使用 setcookie()函数并传入 Cookie 项名称。

【示例 12-4】以下代码演示为存在的 Cookie 项 name 设置空值。

```
<?php
    setcookie('register');          //删除 Cookie 项 register
    print_r($_COOKIE);              //输出全局变量$_COOKIE 信息
?>
```

注意：该示例的运行效果在刷新页面后才可以显示。

代码运行结果如图 12.8 所示。从运行结果可以看出，register 项的值已经为空。使 Cookie 项过期则可以将 setcookie()函数的过期时间设置为当前时间或者当前时间之前的时间。

【示例 12-5】以下代码演示将 Cookie 项 name 设置为过期。

```
<?php
    setcookie('name','',time());        //将过期时间设置为当前时间
    print_r($_COOKIE);                  //输出全局变量$_COOKIE 的信息
?>
```

注意：该示例的运行效果在刷新页面后才可以显示。

代码运行结果如图 12.9 所示。



图 12.8　运行结果



图 12.9　运行结果

从运行结果可以看出，Cookie 项 name 因为过期而不再存在于全局变量$_COOKIE 中。

12.1.5　使用 Cookie 记录登录状态

Cookie 的使用方法在前面都做了比较详细的介绍，下面演示一个简单使用 Cookie 记录用户登录状态并在两个页面间传递。

【示例 12-6】以下代码演示使用 Cookie 记录登录状态，并在两个页面间传递。

a.php 文件如下：

```
<?php
    setcookie('login','true',time()+60*60);     //设置登录状态为 TRUE
    echo '<a href=b.php>跳转到之前的页面</a>';
?>
```

b.php 文件如下：

```
<?php
    error_reporting(NULL);         //关闭错误报告
    if($_COOKIE['login']=='true') //判断 Cookie 中的登录状态并输出相应提示
        echo '你已成功登录该站点，可以进行操作。';
    else{
        echo '你未登录该站点，不允许操作。<br />';
        echo '<a href=a.php>去登录</a>';
    }
?>
```

我们应该首先运行 b.php 文件，会出现如图 12.10 所示的结果。从运行结果可以看出，目前我们没有登录该站点，并可以单击“去登录”链接登录，如图 12.11 所示。在单击“去登录”链接后，浏览器会访问 a.php 文件，该文件会设置 Cookie 并输出跳转链接。单击“跳转到之前的页面”链接，将出现如图 12.12 所示的界面。



图 12.10　运行结果



图 12.11　单击“去登录”链接

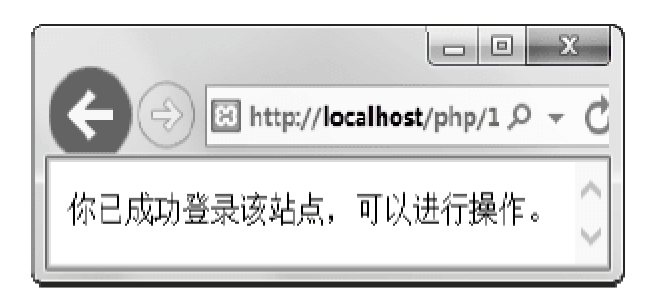


图 12.12　成功登录界面

运行结果提示我们已经成功登录站点，这就是该页面读取到了 Cookie 中设置的 login 项。

12.2　Session 技术

由于 Cookie 文件是保存在客户端磁盘中的，因此相对来说是不安全的，而且目前的浏览器都提供禁用 Cookie 的选项，因此相对来说这是不可控的因素。如果客户将 Cookie 禁用，那么我们 12.1 节中的登录站点示例则会导致用户始终无法登录到站点。Session 技术同 Cookie 技术类似，同样是使用文件来记录一些信息，而最大的不同点就在于 Session 文件是保存在服务器端的。

12.2.1　创建 Session

与 Cookie 不同，如果在 PHP 中需要使用 Session，可以使用 session_start()函数来创建或者读取一个 Session，该函数的语法如下：

```
bool session_start ( void )
```

该函数不接收任何参数。函数执行后会创建或则读取一个 Session 文件。创建 Session 文件时 PHP 会为该文件生成一个唯一的 ID。读取 Session 文件则是获取通过 POST、GET 方式或者 Cookie 传递的 Session 名称和 ID。Session 名称可以通过 session_name()函数获取或者设置，该函数的原型如下：

```
string session_name ([ string $name ] )
```

可选参数用来设置当前 Session 的名称，如果省略该参数则为获取 Session 的名称。Session 默认名称在 php.ini 文件中设置：

```
session.name = PHPSESSID
```

获取 Session 的 ID 可以使用函数 session_id()，该函数的原型如下：

```
string session_id ([ string $id ] )
```

可选参数用来设置当前 Session 的 ID，如果省略该参数则为获取 Session 的 ID。

【示例 12-7】以下代码演示创建一个 Session 并获取该 Session 的名称和 ID。

```
<?php
    session_start();          //创建一个 Session
    echo '当前 Session 的名称为：'.session_name();
    echo '<br />当前 Session 的 ID 为：'.session_id();
?>
```

代码运行结果如图 12.13 所示。从运行结果可以看到当前 Session 的名称和 ID。在前面说过 Session 是存储在服务器端的文件，因此我们可以查看该 Session 文件。Session 文件的保存路径可以在 php.ini 文件中配置：

```
session.save_path =f "D:\xampp\tmp"
```

Session 文件是以 sess_sessionID 的形式保存的。我们可以通过查看 php.ini 中设置的 Session 保存目录查看该文件，示例 12-7 中 Session 文件的属性如图 12.14 所示。



图 12.13　运行结果



图 12.14　示例 12-7 创建的 Session 文件属性

由于我们只是创建了一个 Session，并没有在其中存储信息，因此该文件中没有内容。

12.2.2　设置、读取和删除 Session

在 Session 被创建以后就可以对 Session 进行设置、读取和删除操作。

1. 设置 Session

设置 Session 也就是在 Session 文件中存储内容。Session 文件的内容是保存在全局变量 $_SESSION 中的。该变量也是一个数组，信息可以以键值对的形式作为数组元素被存储。向 Session 中存储数据则要比向 Cookie 中存储数据简单。操作过程类似于为数据添加或者

删除元素。

【示例 12-8】以下代码演示向 Session 中存储内容。

```
<?php
    session_start();            //创建 Session
    $_SESSION['name']='Ken';    //向 Session 中写入内容
    print_r($_SESSION);         //输出 Session 中的内容
?>
```

代码运行结果如图 12.15 所示。从运行结果可以看到 Session 中已经存储了内容。我们可以查看该 Session 文件中的内容，如图 12.16 所示。

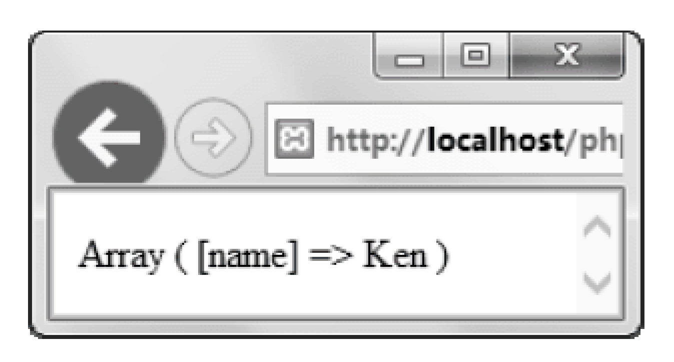


图 12.15　运行结果



图 12.16　示例 12-8 创建的 Session 文件内容

从图 12.16 中可以看到，Session 文件中存储的信息可读性还是比较强的。它是以如下的格式存储的：

```
项名称|值类型:值长度:值
```

删除 Session 中的内容也比较容易。删除其中的一项可以使用 unset()函数释放$_SESSION 中的元素。也可以为$_SESSION 赋值一个空数组或者使用 session_unset()函数来删除 Session 中的所有数据。session_unset()函数的原型如下：

```
void session_unset ( void )
```

删除 Session 中的内容比较简单，这里就不做详细介绍。

2. 读取 Session 内容

获取 Session 中的内容类似于获取 Cookie 中的内容，通常使用如下形式：

```
$_SESSION[名称]
```

该知识点比较简单，这里就不做详细介绍。

3. 删除 Session

删除 Session 同删除 Session 中的内容是不同的。删除 Session 会将 Session 文件删除而不只是其中的数据。删除 Session 可以使用 session_destroy()函数，它的原型如下：

```
bool session_destroy ( void )
```

该函数不接收任何参数。需要注意的是 Session 文件会在一定的时间后自动被删除，这个时间可以在 php.ini 文件中设置：

```
session.gc_maxlifetime = 1440
```

上面的配置就说明，Session 的最大存在的时间就为 1440 秒。

【示例 12-9】以下代码演示创建并且删除一个 Session。

```
01  <?php
02      session_start();          //创建一个 Session
03      //获取 Session 名称和 ID
04      echo '当前 Session 的名称为：'.session_name();
05      echo '<br />当前 Session 的 ID 为：'.session_id();
06      session_destroy();        //删除 Session
07  ?>
```

代码运行结果如图 12.17 所示。



图 12.17　运行结果

运行结果获取到了 Session 的名称和 ID，则说明 Session 被成功创建。而代码中第 6 行则删除了该 Session 文件，因此我们并不会在存放 Session 文件的目录中找到“sess_p9mn9th420hrsljnekju4oto11”文件。

12.2.3　使用 Session 记录信息

Cookie 文件是存储在客户端的，是不安全的，因而不可以将一些敏感的数据存放在 Cookie 文件中。而 Session 存放在服务器端，相对来说是比较安全的。我们可以在 Session 中存放一些比较敏感的数据。

1. Session 与 Cookie 联合记录敏感信息

Session 可以通过 Cookie 传输的名称和 ID 取回已经存在的 Session。因此我们需要将 Session 的名称和 ID 存入 Cookie，通常的形式如下：

```
setcookie(session_name(),session_id(),过期时间)
```

【示例 12-10】以下代码演示使用 Session 保存用户名和密码并在两个页面间传递。

a.php 文件如下：

```
a.php 文件
<?php
    session_start();                    //创建 Session
    setcookie(session_name(),session_id(),time()+60*60);
                                        //使用 Cookie 保存 Session 名称和 ID
    //设置 Session 内容
    $_SESSION['user']='Anne';
    $_SESSION['password']='mypassword';
    echo '<a href=b.php>登录</a>';     //输出登录链接
?>
```

b.php 文件如下：

```
b.php 文件
<?php
    session_start();                    //取回 Session
    //判断用户名和密码是否正确并输出对应的欢迎信息
    if($_SESSION['user']=='Anne'&&$_SESSION['password']=='mypassword')
        echo "欢迎{$_SESSION['user']}回来~~~";
    else
        echo "欢迎游客光临本站~~~";
```

```
?>
```

我们应该首先运行 a.php 文件，会出现如图 12.18 所示的界面。由于 a.php 中的用户名和密码都与 b.php 文件中的用户名和密码一致，因此可以作为注册用户登录，如图 12.19 所示。

下面我们将 a.php 文件中的用户名或者密码修改为同 b.php 文件中用户名和密码不对应后，再次运行 a.php 界面，然后单击“登录”链接，会出现如图 12.20 所示的界面。



图 12.18　登录界面



图 12.19　注册用户登录



图 12.20　游客登录

经过以上的示例可以得知，Session 已经很好地完成了我们期望的工作。我们知道浏览器提供了禁用 Cookie 的选项，可以在 IE 浏览器的 Internet 选项中进行设置，如图 12.21 所示。通过如图 12.21 所示的设置禁用 Cookie 后，将示例中两个文件中的用户名和密码设置为一致后再次执行 a.php 文件并单击“登录”链接，会出现如图 12.22 所示的界面。



图 12.21　Cookie 设置



图 12.22　运行结果

该结果显示我们现在是以游客身份登录的，并且有一条 user 变量未定义的提示信息。这就是因为 Cookie 被禁用而导致 a.php 文件创建的 Session 不能被 b.php 文件取回。

2. 以重写 URL 方式传送 Session 名称和 ID

在禁用 Cookie 后，我们可以看到依赖 Cookie 的示例 12-10 就不能很好地工作了。Session 需要通过 Session 的名称和 ID 来取回存在的 Session 信息。那么我们就可以通过在链接中显式地传递这些信息以供 Session 取回正确的 Session 信息。在地址中显式地传递信息可以为以下的形式：

```
URL 地址?参数
```

将 Session 的名称和 ID 作为参数：

```
URL 地址?session_name()=session_id()
```

【示例 12-11】以下代码演示将示例 12-10 改写为不依赖 Cookie 工作的站点。

a.php 文件如下：

```
<?php
```

```
    session_start();                        //创建 Session
    $sid=session_name().'='.session_id();   //将 Session 名称和 ID 链接
    //设置 Session 内容
    $_SESSION['user']='Anne';
    $_SESSION['password']='mypassword';
    echo "<a href=b.php?{$sid}>登录</a>";    //输出登录链接
?>
```

b.php 文件如下：

```
<?php
    session_start();                        //取回 Session
    //判断用户名和密码是否正确并输出对应的欢迎信息
    if($_SESSION['user']=='Anne'&&$_SESSION['password']==
        'mypassword')
        echo "欢迎{$_SESSION['user']}回来~~~";
    else
        echo "欢迎游客光临本站~~~";
?>
```

我们同样首先运行 a.php 文件，会出现如图 12.23 所示的界面。单击“登录”链接，会出现如图 12.24 所示的界面。



图 12.23　登录界面



图 12.24　登录成功

运行结果与我们的期望是一致的，由此我们就掌握了不依赖 Cookie 来记录信息的方法。由于类似于示例 12-11 的使用方式很常见，而这种方式最大的不同就是显式地以如下的形式传递 Session 的信息：

```
session_name().'='.session_id()
```

因此 PHP 将以上语句的值定义为常量 SID，因此我们就可以将 a.php 文件改写如下：

```
<?php
    session_start();                        //创建 Session
    $sid=SID;                               //获取 Session 名称和 ID
    //设置 Session 内容
    $_SESSION['user']='Anne';
    $_SESSION['password']='mypassword';
    echo "<a href=b.php?{$sid}>登录</a>";   //输出登录链接
?>
```

代码执行效果同示例 12-11 中 a.php 文件的执行效果是一致的，这里就不再演示。

12.3　小　　结

本章我们学习了 Web 中非常重要的 Cookie 和 Session 技术。如果没有这种技术，Web 的很大一部分功能都会缺失。因此，掌握本章的所有内容是完全有必要的。Cookie 和 Session

技术的难点不在于如何在 PHP 中使用，而在于其工作的原理，因此读者应该仔细阅读并掌握 Cookie 和 Session 原理部分的知识。

12.4 本章习题

1. 设置一个 Cookie 变量保存用户访问该页面的时间，然后在一个新页面中读取该时间。

2. 使用设置 Cookie 保存 Session 的 ID 和名称信息，并在第二个页面获取该 ID 和名称。

3. 设置一个 Session 变量保存用户访问该页面的时间，然后在一个新页面中读取该时间。

4. 在 Session 变量中保存用户名 user 和密码 password。然后，在新页面中验证这些信息，如果匹配则显示欢迎信息，否则提示信息不匹配。

第 3 篇　PHP 开发实战

第 13 章　网 站 模 版

在现在很多网站的建设中，都很少直接从零开始搭建，而是会选择一些网站模版，然后定制一些自己的内容后就可以投入使用。本章我们不会介绍如何定制网站模版而是主要介绍 Discuz!和 Joomla!网站模版的搭建。

13.1　搭建 Discuz!论坛

Discuz!是在国内使用量非常大的一个社区论坛程序。该程序是由康盛创想科技有限公司编写和维护。Discuz!的源代码可以免费下载，但是它并不是开源软件。本节我们就来详细介绍 Discuz!的安装方法以及简单的使用方法。

13.1.1　Discuz!安装

Discuz!的源代码可以通过访问“http://www.comsenz.com/downloads/install/discuzx”来获取，目前最新的版本是 X2.5，如图 13.1 所示。

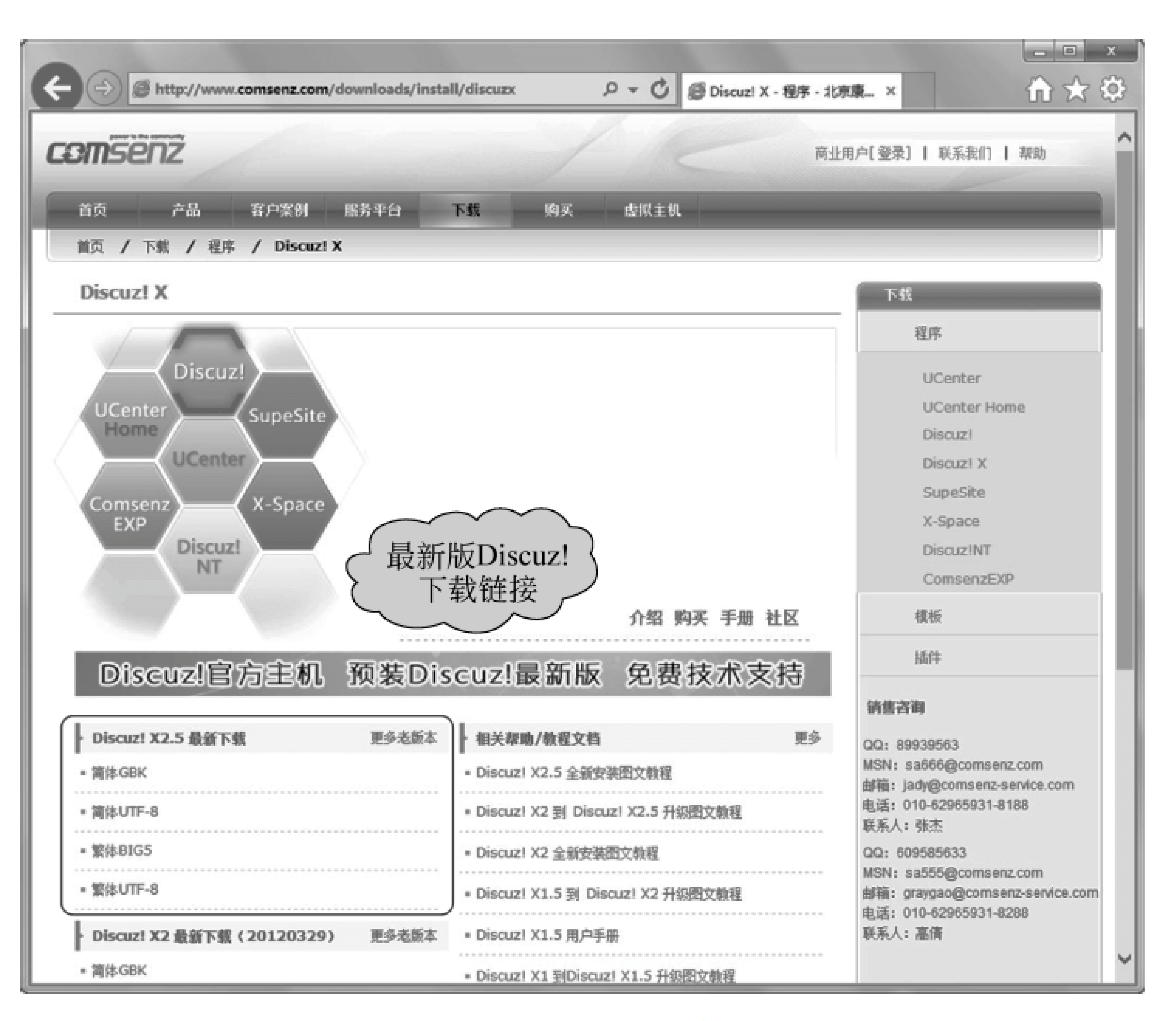


图 13.1　Discuz!下载页面

Discuz!提供多种编码的源文件，这里我们使用 UTF-8 编码的源文件。下载的源文件名为 Discuz_X2.5_SC_UTF8.zip 的 zip 压缩格式。使用解压缩工具打开它可以看到 readme、upload、utility 这 3 个文件夹，如图 13.2 所示。



图 13.2　压缩包中的文件

其中的 readme 文件夹中存放一些相关的文档；upload 文件夹存放 Discuz!软件的源代码；utility 文件夹中存放 Discuz 的升级工具。我们在安装 Discuz!的时候只需要使用到 upload 文件夹，因此我们需要将该文件夹解压缩到 Web 服务器目录下，为了便于记忆，我们可以将它重命名为 Discuz。前期准备已经就绪。Discuz!的安装是非常简单的，我们只需要在浏览器中访问服务器目录下的 Discuz 文件夹即可，如图 13.3 所示。



图 13.3　Discuz!授权协议

（1）首先出现的是 Discuz!的中文版授权协议。在阅读后，单击 “我同意”按钮，进行下一步操作，如图 13.4 和图 13.5 所示。



图 13.4　检查安装环境

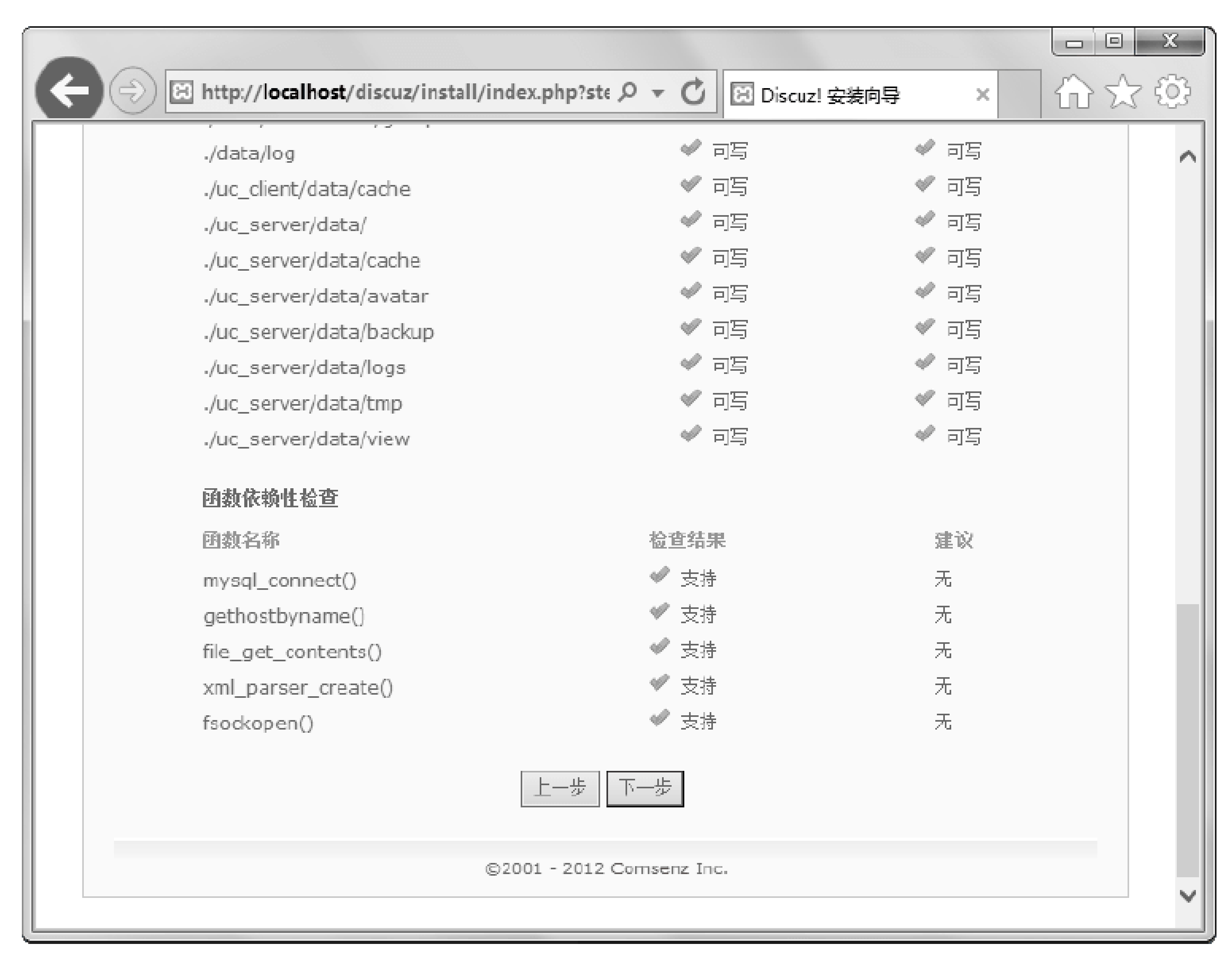


图 13.5　检查安装环境

（2）该步骤用来检测当前环境是否可以正确安装 Discuz!。在 Windows 操作系统中通常不会有太大的问题。在 UNIX 或者类 UNIX 系统中可能会出现没有目录操作权限的问题。当检测结果均符合安装要求时，单击“下一步”按钮继续，如图 13.6 所示。



图 13.6　设置运行环境

（3）该步骤让我们选择安装 Discuz!的方式，这里我们选择全新安装模式，然后单击“下一步”按钮继续，如图 13.7 所示。

（4）该步骤用来设置 Discuz!安装所依赖的数据库，我们需要正确填写数据库的用户名和密码，并设置一个网站管理员的账号和密码，然后单击“下一步”按钮继续安装，如图 13.8 所示。

（5）该步骤会显示创建依赖的数据库的状态，无须我们进行任何操作。在数据库创建完成后，会出现如图 13.9 所示的提示界面。

图 13.7　安装数据库



图 13.8　创建数据库



图 13.9　安装成功后的界面

（6）在该界面中会向我们推荐一些服务，这里我们选择暂不开通即可跳过。跳过推荐的服务后，浏览器会跳转至安装完成的 Discuz!主页面，如图 13.10 所示。

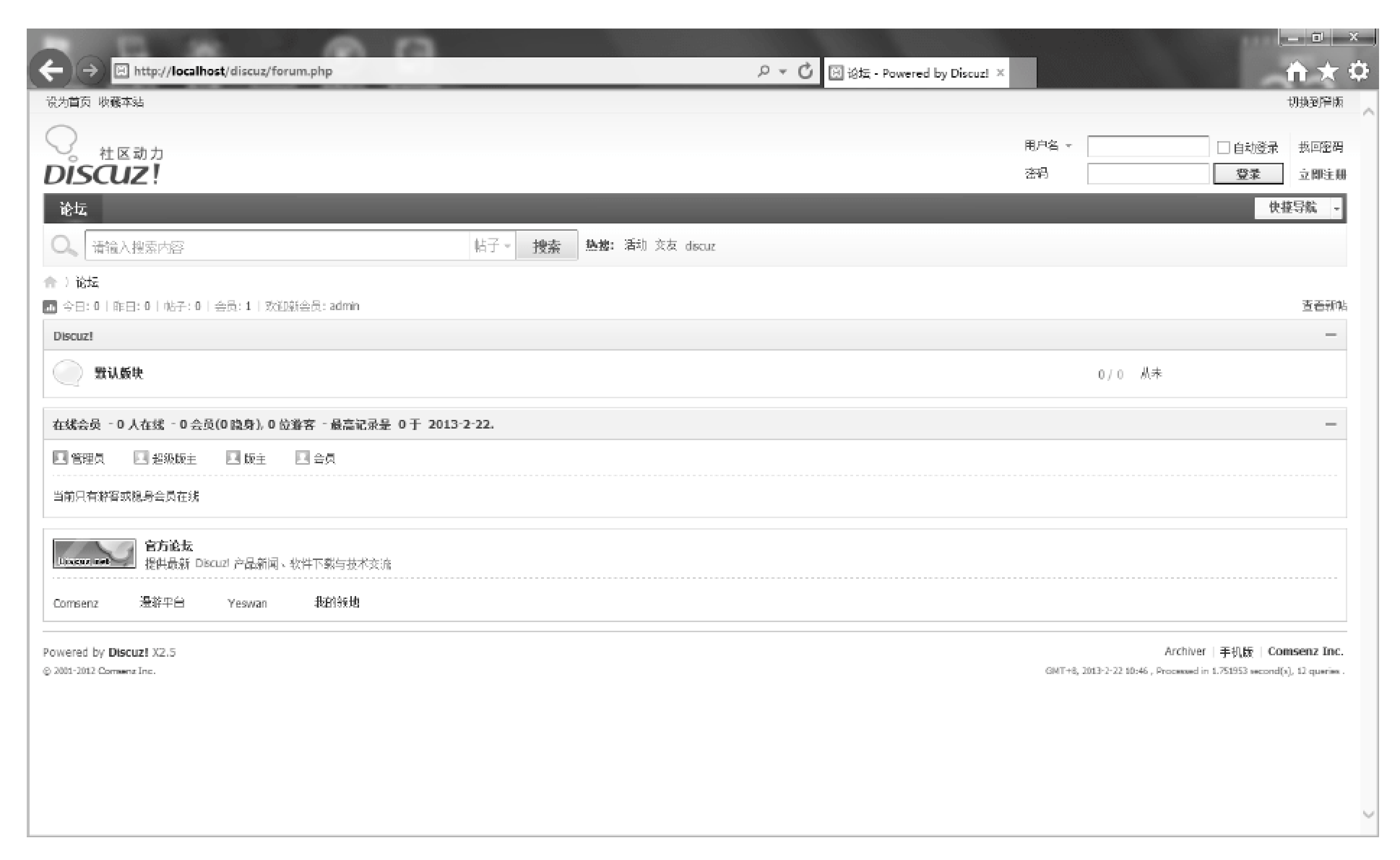


图 13.10　Discuz 主页面

到此为止，Discuz!程序就安装完毕了，我们还是可以感觉到安装过程是非常智能的，几乎不需要我们进行太多的操作。

13.1.2　登录站点

在 Discuz!安装完成后，我们只需要访问 http://localhost/discuz/forum.php 即可进入 Discuz!的主页面。管理员登录站点的方式同普通用户相同，即为从图 13.11 所示的图中登录。



图 13.11　管理员登录

成功登录后的界面如图 13.12 所示。

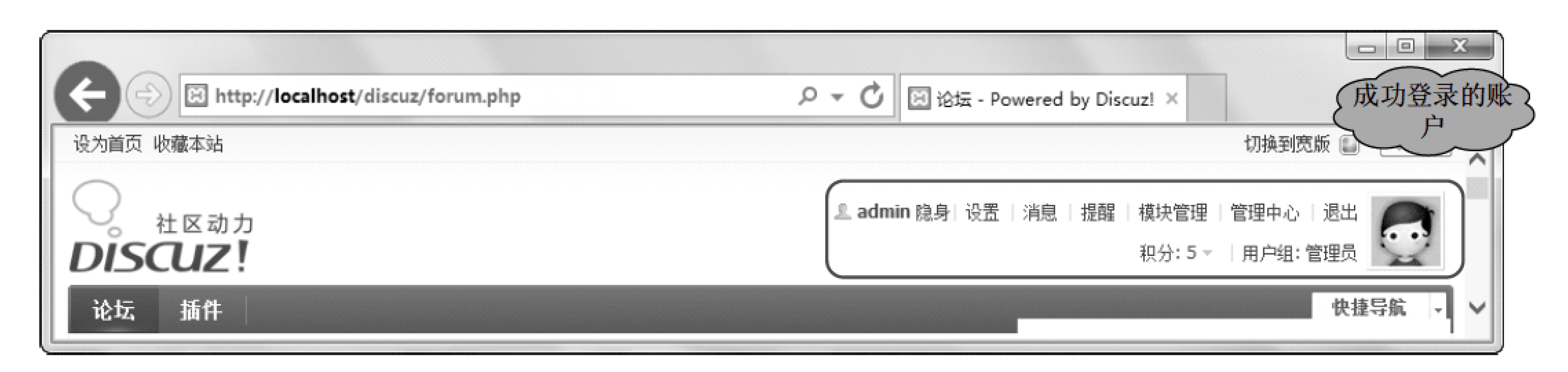


图 13.12　登录后的界面

在登录后，即可对站点进行全方位的管理。我们可以单击用户头像左侧的“管理中心”链接，进入管理中心，如图 13.13 所示。



图 13.13　管理中心

13.1.3　管理站点信息

站点信息是对站点最直观的展现，例如，站点名称就可以体现出该网站的类型。站点信息可以在全局类中的站点信息中修改，如图 13.14 所示。

站点信息中可以修改的内容都对应了其说明，这里我们将站点名称和网站名称均改为“PHP 技术网”并提交修改。我们返回到首页面应该可以看到两处改变，如图 13.15 所示。



图 13.14　管理站点信息



图 13.15　设置的站点信息

从图 13.15 中标出的两处位置可以看到，它们均改为我们前面设置的信息。当然，网站的 LOGO 也是可以改变的，本书中不做介绍。

13.1.4　管理版块

管理版块是论坛程序中最常用的管理项，它可以在管理中心的论坛类和版块管理项中进行设置，如图 13.16 所示。



图 13.16　版块管理

从图 13.16 中可以得知，当前论坛有一个分区“Discuz!”，该分区下有一个版块“默认版块”。下面就来演示添加一个新分区“VIP 专区”并在其下添加“学习”和“交流”两个版块。操作过程如图 13.17 所示。



图 13.17　添加新分区和新版块

在提交以上修改之后，返回首页面可以查看修改后的效果，如图 13.18 所示。

当然，在管理中心可以对版块进行设置版主以及设置权限等等非常详细的操作。这里就不做详细介绍。



图 13.18　添加的新分区和新版块

13.1.5　添加插件

插件可以用来对网站功能进行扩展，以提高网站的质量。Discuz!有内置的应用中心，我们可以在其中选择一些插件进行安装使用。下面就来简单介绍一款签到插件的安装和使用。我们可以在应用类的应用中心项中查找“每日签到”关键字来找到该应用，如图 13.19 所示。



图 13.19　查找应用

找到以上应用并安装，然后我们可以在应用类中的插件项中启用它，如图 13.20 所示。在启用该插件后，我们可以在站点首页面中看到它，如图 13.21 所示。



图 13.20　插件信息



图 13.21　签到插件

打开每日签到，会出现如图 13.22 所示的页面，用户可以在其中发表心情以及查看签到排行榜。



图 13.22　签到页面

从上面安装插件的示例中我们也可以感受到，为 Discuz!扩展一些功能是非常方便的。其他相关插件的使用同签到插件类似，这里不做过多介绍，读者可以自行试用。

13.2　搭建 Joomla!站点

Joomla!是一套以 PHP 撰写的自由、开放源代码的内容管理系统。相对 Discuz!来说，Joomla!更加侧重于创建个人网站。由于其使用 GPL 授权，因此在全球范围内使用非常广泛。本节就来介绍 Joomla!系统的安装及简单的使用方法。

13.2.1　Joomla!安装

Joomla!的源代码包可以从站点 http://www.joomla.org/download.html#j3 下载，如图 13.23 所示。



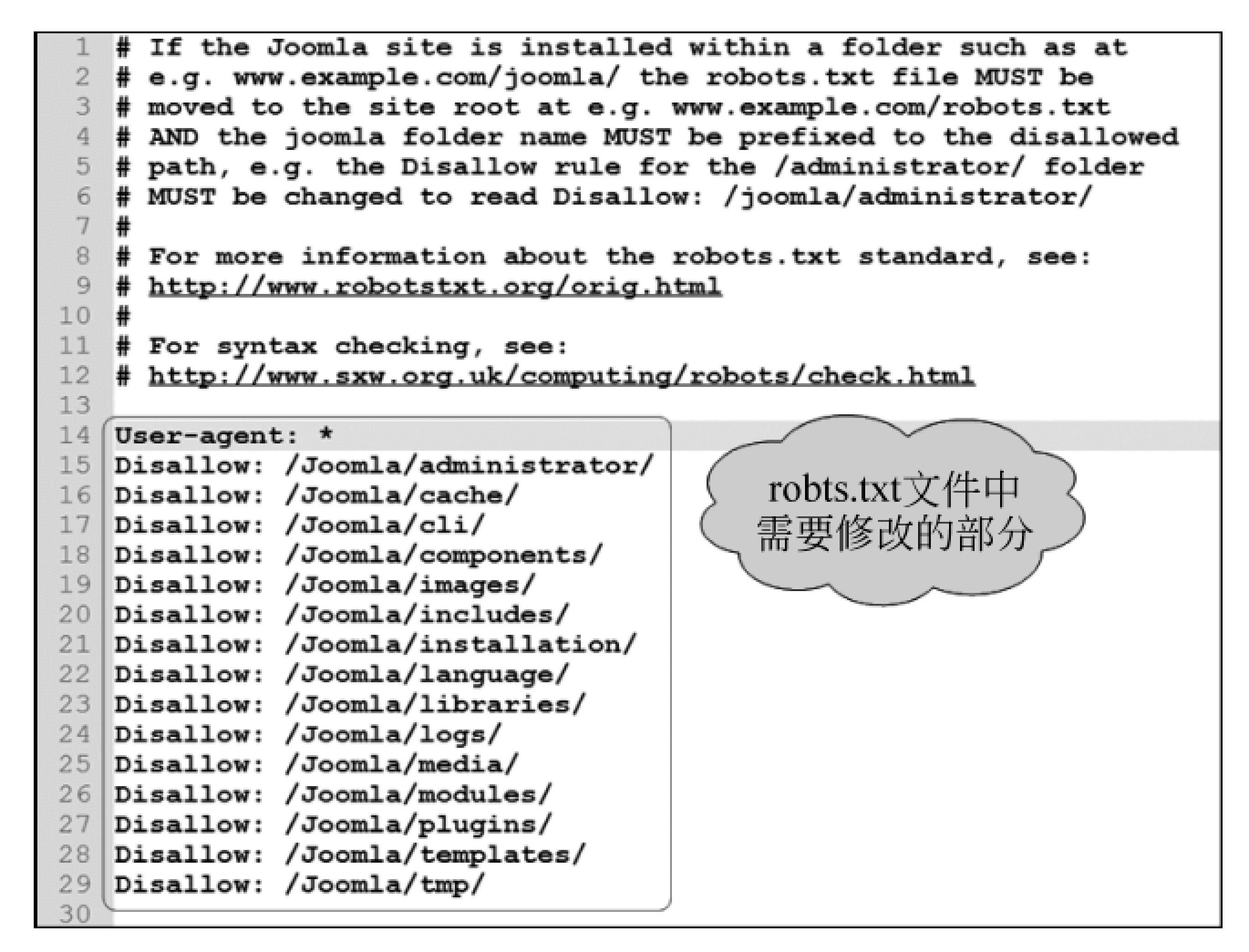
图 13.23　下载 Joomla!源代码

当前最新版本为 3.0.3，源文件为一个名为“Joomla_3.0.3-Stable-Full_Package.zip”的 zip 压缩包。为了便于讲解，我们需要将其解压缩到一个文件夹（例如 Joomla）中并将该文件夹复制到 Web 服务器根目录下，如图 13.24 所示。



图 13.24　Joomla!源代码位置

由于 Joomla 的默认设置是安装在 Web 服务器根目录下的，而我们将其源文件放入一个文件夹中，因此，对其中的 robots.txt 文件进行配置。需要对 robots.txt 文件进行的配置就是将文件中 Disallow 后的路径均改为相对于 Web 服务器根目录的路径，如图 13.25 所示。



```
# If the Joomla site is installed within a folder such as at
# e.g. www.example.com/joomla/ the robots.txt file MUST be
# moved to the site root at e.g. www.example.com/robots.txt
# AND the joomla folder name MUST be prefixed to the disallowed
# path, e.g. the Disallow rule for the /administrator/ folder
# MUST be changed to read Disallow: /joomla/administrator/
#
# For more information about the robots.txt standard, see:
# http://www.robotstxt.org/orig.html
#
# For syntax checking, see:
# http://www.sxw.org.uk/computing/robots/check.html

User-agent: *
Disallow: /Joomla/administrator/
Disallow: /Joomla/cache/
Disallow: /Joomla/cli/
Disallow: /Joomla/components/
Disallow: /Joomla/images/
Disallow: /Joomla/includes/
Disallow: /Joomla/installation/
Disallow: /Joomla/language/
Disallow: /Joomla/libraries/
Disallow: /Joomla/logs/
Disallow: /Joomla/media/
Disallow: /Joomla/modules/
Disallow: /Joomla/plugins/
Disallow: /Joomla/templates/
Disallow: /Joomla/tmp/
```


图 13.25　修改后的 robots.txt 文件

修改 robots.txt 文件后将该文件剪切到服务器的根目录下，如图 13.26 所示。



图 13.26　剪切 robots.txt 文件到 Web 服务器根目录

到此为止，安装前的准备就做好了。同安装 Discuz!类似，我们通过在浏览器中访问 http://localhost/joomla 即可开始安装 Joomla!。首先出现的是配置设置，如图 13.27 所示。

（1）在设置好网站的名称、管理员的邮箱和账户信息后，单击“下一步”按钮，将会出现如图 13.28 所示的界面。

（2）数据库设置界面需要我们设置数据库类型、Joomla!站点使用的数据库名称、数据表前缀及登录数据库的用户名和密码等信息。在正确设置这些信息后，就可以单击“下一步”按钮继续，将会出现如图 13.29 和图 13.30 所示的界面。



图 13.27　主要配置设置



图 13.28　数据库设置界面



图 13.29　配置信息界面



图 13.30　配置信息界面

（3）在该界面中列出了一些安装前的配置信息，以及询问是否安装示范数据。为了便于学习，这里选择“不安装示范数据”。在确认了配置信息无误后，我们就可以单击“安装”按钮进行安装，如图 13.31 所示。



图 13.31　安装过程

（4）在安装操作执行完毕后，将会出现如图 13.32 所示的界面。



图 13.32　安装完成界面

Joomla!默认只安装有 English 语言，我们可以通过单击界面中的“增补步骤：安装语言”按钮（确认这个按钮，是否在界面上）来安装“简体中文”语言，如图 13.33 所示。



图 13.33　安装语言

（5）在该界面中，我们选择 Chinese Simplified 语言，并单击“下一步”按钮。系统就会下载并安装选定的语言，语言安装完成后需要选择默认的安装语言，如图 13.34 所示。

（6）在该步骤中，我们将前台和后台默认语言均选为“中文（简体）”，并单击“下一步”按钮完成安装，如图 13.35 所示。



图 13.34　选择安装语言



图 13.35　完成安装

（7）在安装完成的界面，安装程序会提示我们删除安装目录，这是 Joomla!的一项安全措施。单击“删除安装目录”按钮后，将出现如图 13.36 所示的界面。



图 13.36　删除安装目录后的界面

现在我们就可以单击“网站”按钮来进入站点，如图 13.37 所示。

虽然前台语言已经被设置为简体中文，但是一些默认的数据本身就是英文的，因此一些数据内容仍然是以英文显示的。

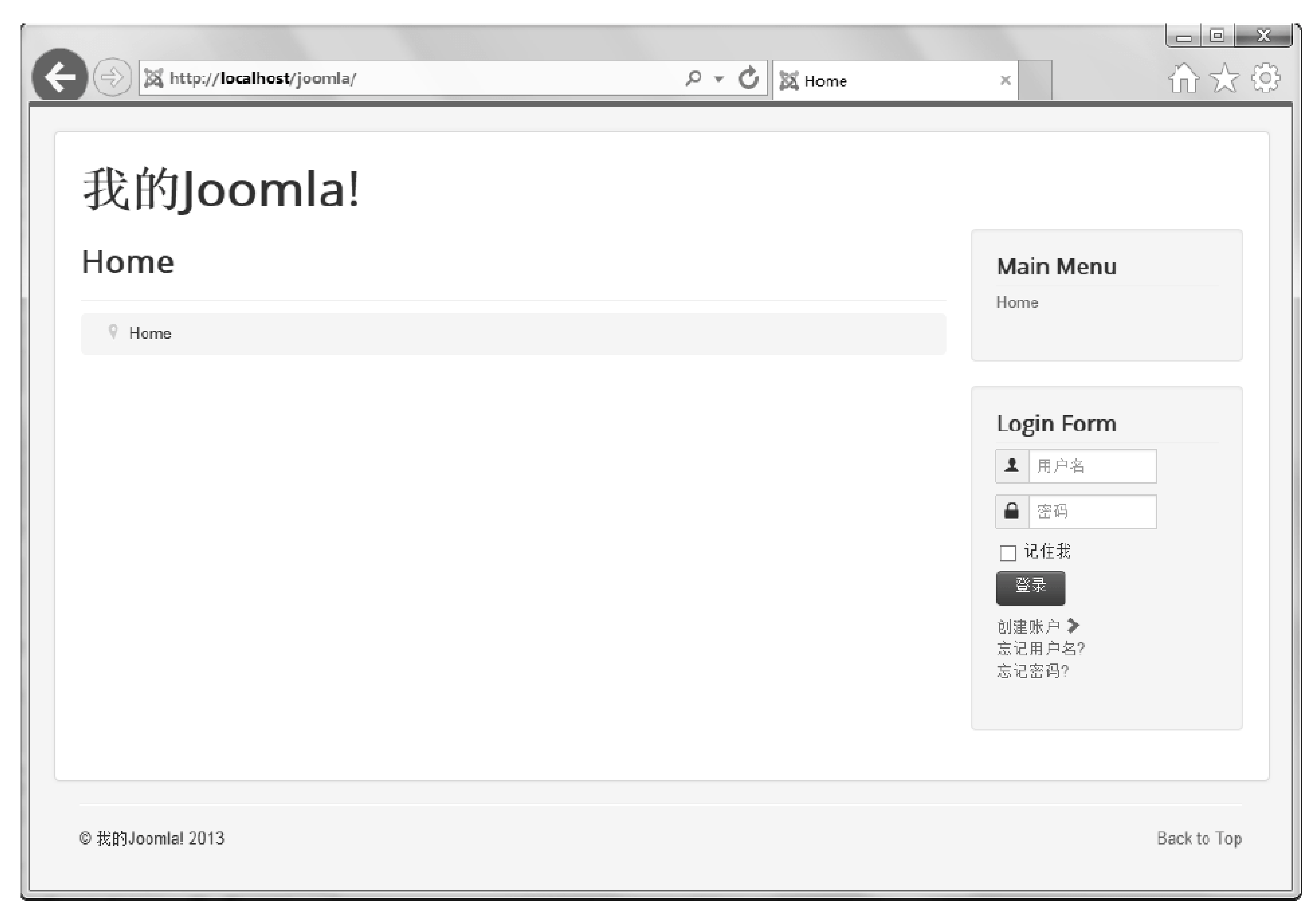


图 13.37　站点首页

13.2.2　管理 Joomla!

管理 Joomla!需要登录后台管理。登录后台管理需要访问“http://localhost/joomla/administrator/”，如图 13.38 所示。

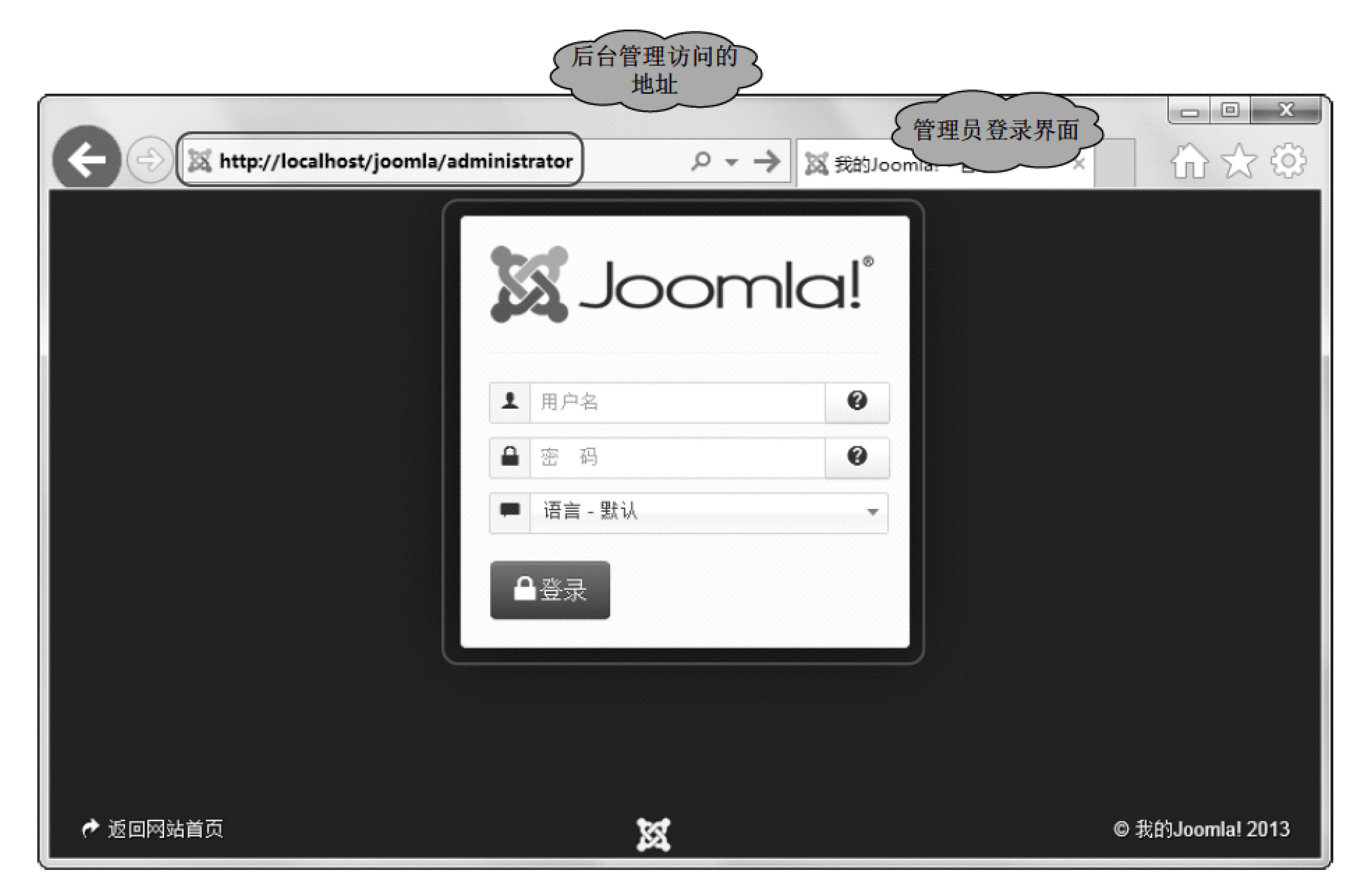


图 13.38　后台管理登录界面

登录后的界面如图 13.39 所示。

Joomla!的后台管理也是非常明了的，读者可以分别查看每个选项，熟悉可以管理的项目。



图 13.39　后台管理界面

13.2.3　Joomla!使用

Joomla!的后台管理与 Discuz!没有太大的不同，本节就简单介绍一下如何发表文章以及如何为 Joomla!安装插件。

1. 添加文章

添加文章是 Joomla!中常用的操作。发表文章既可以在控制面板页中的快捷命令中选择“添加文章”，也可以在内容类的“文章管理”中选择“添加文章”，如图 13.40 所示。



图 13.40　为站点添加文章

在使用任意一种方式选择“添加文章”项后，均会出现如图 13.41 所示的文章编辑页面。

下面为文章编写标题和内容，并且选择一个分类（添加分类同添加文章类似，这里不再详细介绍，然后将文章在头版显示，如图 13.42 所示。

编辑完成后可以将文章保存，然后单击页面左下角的“浏览网站”来查看效果，如图 13.43 所示。

从图 13.43 中可以看到文章已经被正确显示到了站点中，当然我们可以为文章进行添加图片、链接或者改变字体等操作，读者可以自行尝试。



图 13.41　添加文章页



图 13.42　编辑文章



图 13.43　添加的文章

2. 扩展 Joomla!

Joomla!的扩展可以通过后台管理的“帮助”项中选择“Joomla 扩展”或者访问“http://extensions.joomla.org/”来获取扩展资源，如图 13.44 所示。

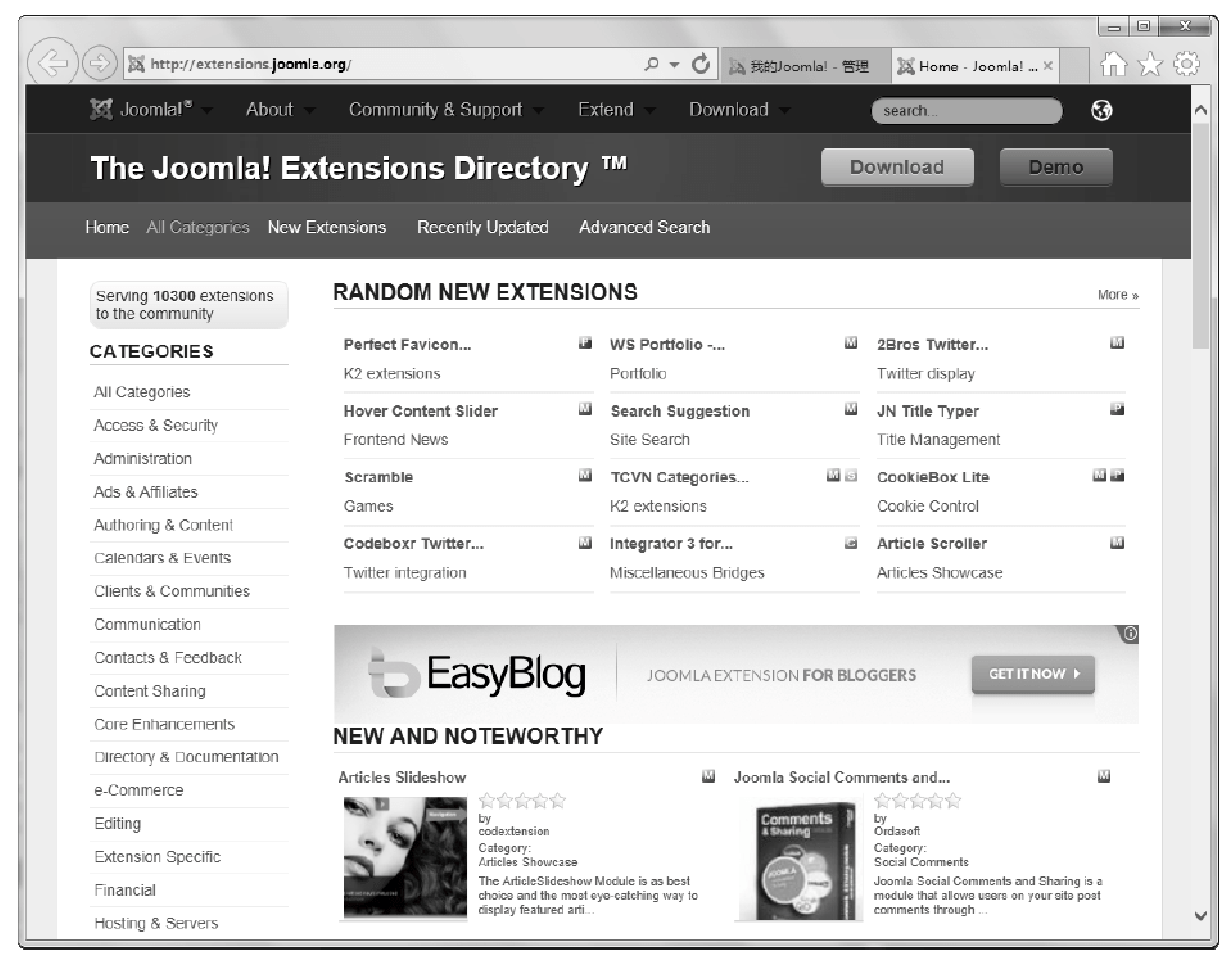


图 13.44　Joomla!扩展资源站点

Joomla!有非常多的扩展资源可以使用，这里就来演示添加一个名为 Googlemaps Plugin 地图插件。我们首先需要下载这个插件的安装包，如图 13.45 所示。

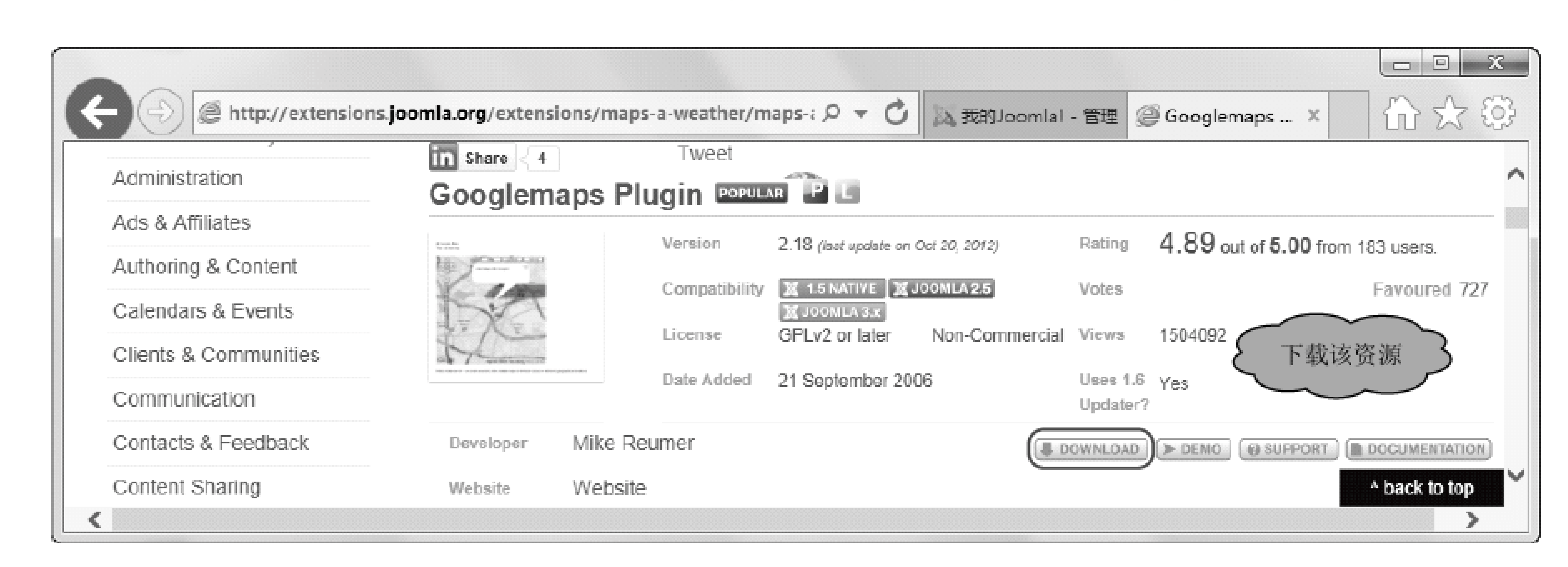


图 13.45　下载插件安装包

Joomla!的插件安装包通常是以 ZIP 文件格式发布的，当前最新版本下载后的文件名为 plugin_googlemap_J25_J30.2.18.zip。Joomla!安装扩展也非常简单，可以通过选择“扩展”类的“扩展管理”来打开扩展安装界面，如图 13.46 所示。

在该页面中浏览刚才下载的文件，并且单击“上传并安装”按钮完成安装，如图 13.47 所示。

在插件安装成功后我们可以管理插件，如图 13.48 所示。

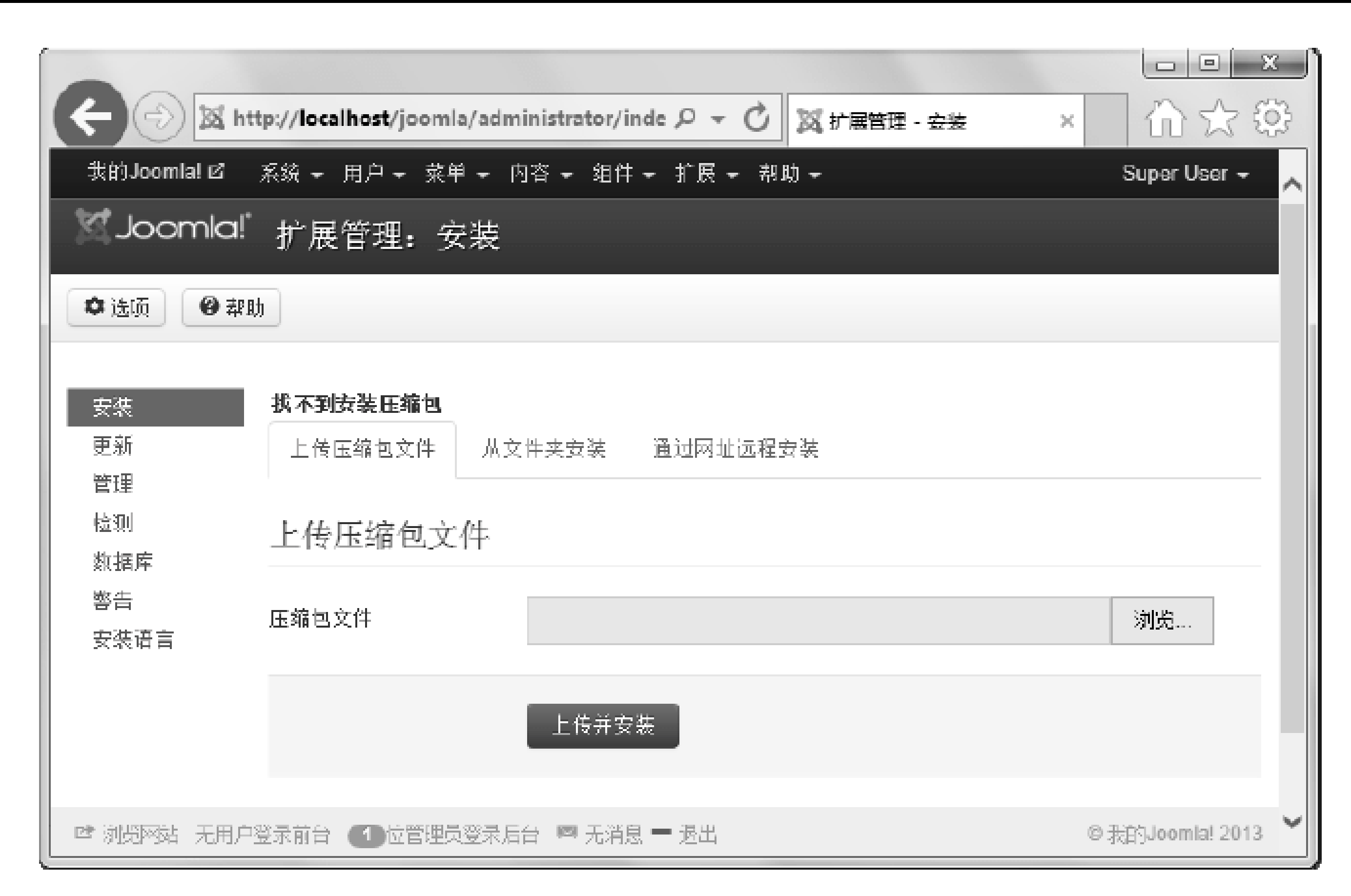


图 13.46　安装扩展界面



图 13.47　扩展安装成功



图 13.48　管理插件

在插件被启用后，我们就可以使用它了，下面我们就来演示在文件中使用 Google 地图显示位置，如图 13.49 所示。



图 13.49　使用地图插件

保存文章后我们再次浏览站点首页，如图 13.50 所示。



图 13.50　使用地图插件

可以从图 13.50 中看到，我们在文章中很简单地使用 Google 地图标注了位置。到这里为止，插件的安装以及使用的知识就介绍完毕了，读者也可以自己尝试使用其他的插件。

13.3　小　　结

本章中主要介绍了两个典型的网站模版 Discuz!和 Joomla!。使用 Discuz!可以非常快速地搭建起一个可以使用的论坛；Joomla!则是快速创建一个个人网站的完美选择。至于这些站点的后台管理，读者大可以按照自己的意愿来尝试各种扩展及设置，这是学习后台管理

的一个好方式。

13.4 本章习题

1. 搭建 Discuz!论坛，并设置网站名称为“我的第一个 PHP 论坛站点”。
2. 在新搭建的 Discuz!论坛中新建“交易”版块。
3. 搭建一个 Joomla!站点并将站点名称设置为“美食世界”。
4. 在新搭建的 Joomla!站点首页发布一篇文章并搭配一幅美食的图片。

第 14 章　常 用 模 块

作为本书最后的章节，在这里将会使用我们前面章节介绍的知识来创建一个简易的网盘和一个简单的投票系统，在复习知识的情况下做出一个可以使用的程序。

14.1　小 小 网 盘

网盘是近几年来兴起的一类 Web 应用，用户可以将自己的文件保存在网盘中，然后在任何有网络的环境下管理自己的文件。在本节中，我们自己来写出一个可以使用的简易网盘程序。

14.1.1　功能设计

在一个项目开始之前，通常需要很多事情要做，但是我们这里只是实现一个非常简易的程序，不必使用太专业的软件构建知识。下面我们就以文字来简单描述将要创建的网盘需要的功能。

1. 注册与登录

注册与登录作为简易网盘的主页面，提供注册账户和登录账户的功能，其简易界面效果如图 14.1 所示。已注册用户可以单击“登录”链接，进入登录页面，如图 14.2 所示。登录页面需要判断密码的正确性，如果正确则跳转到网盘管理界面，否则提示重新输入正确的用户名和密码直到成功登录。新用户则可以单击“注册”链接，用户进入注册页面，如图 14.3 所示。



图 14.1　注册与登录界面



图 14.2　登录页面



图 14.3　注册页面

用户提供用户名和密码即可注册。在注册成功后即提示用户进入登录界面，否则提示用户输入正确的用户名和密码直到成功注册。

2. 管理网盘中的文件

在用户成功登录后就进入网盘文件管理页面，如图 14.4 所示。在文件管理界面，每个文件应该提供下载按钮。每个文件夹以链接的形式显示以便进入子文件夹，并且不提供下

载链接。进入子文件夹后应该显示返回上一级链接，如图 14.5 所示。

在每个目录下应该提供上传选项用来上传新文件到当前目录，如图 14.6 所示。





图 14.4　文件管理界面



图 14.5　子目录中的返回上一级链接



图 14.6　上传文件

文件上传后应该立即刷新当前页面以便用户管理。

3. 数据库设计

在简易网盘程序中，需要在注册和登录的时候使用数据库，而我们提供的注册信息只有简单的用户名和密码，因此只需要创建一个只有用户名和密码的简单数据表。SQL 命令如下：

```
CREATE TABLE phppan (
    username VARCHAR(20) NOT NULL,
    password VARCHAR(20) NOT NULL
);
```

创建后的结构表数据如下：

```
mysql> DESCRIBE phppan;
+-----------+-------------+-------+------+-----------+-----------+
| Field     | Type        | Null  | Key  | Default   | Extra     |
+-----------+-------------+-------+------+-----------+-----------+
| username  | varchar(20) | NO    |      | NULL      |           |
| password  | varchar(20) | NO    |      | NULL      |           |
+-----------+-------------+-------+------+-----------+-----------+
2 rows in set (0.01 sec)
```

14.1.2　具体代码实现

本节用来将 14.1.1 节中的设计功能以代码的方式实现。

1. 主页面实现

主页面是网盘程序的入口，为新用户提供注册的链接，为老用户提供登录的链接。实现的 HTML 代码如下：

```
<!--mainpage.html-->
<html>
    <h1>欢迎光临 PHP 网盘</h1><br />
    如果您为新用户，请<a href="register.html">注册</a><br />
    如果您拥有一个账号，请<a href="login.html">登录</a>
</html>
```

该 HTML 代码用来将用户引入对应的界面，只需提供两个跳转链接不需要 PHP 代码。

2. 注册与登录实现代码

注册的实现会有很多的特殊情况，例如用户名或者密码为空、用户名已经存在等。登录则只需判断用户名和密码都不为空，而且与数据库中的数据匹配。

（1）注册实现

首先，我们应该创建一个 HTML 页面，然后将注册信息传递给 PHP 页面。HTML 页面的实现代码如下：

```
<!--register.html->
<html>
    <h1>注册网盘</h1>
    <form action="register.php" method="post">
        用户名：
        <input type="text" name="username">
        <br /><br />
        密    码：
        <input type="text" name="password">
        <br />
        <input type="submit" value="提交注册">
    </form>
</html>
```

PHP 代码的实现如下：

```
/*register.php*/
<?php
    error_reporting(NULL);                    //关闭错误信息显示
    $link=mysql_connect('localhost','root','')or die("数据库连接失败！");
                                              //连接数据库
    mysql_select_db('mydatabase',$link);     //选择数据表
    $username=$_POST['username'];             //获取 HTML 中的用户名
    $psw=$_POST['password'];                  //获取 HTML 中的密码
    $sql="SELECT * FROM phppan WHERE username='{$username}'";
    $res=mysql_query($sql,$link)or die("查询失败".mysql_error($link));
                                              //查询用户名是否被注册
    $arr=mysql_num_rows($res);                //返回查询结果
    if(empty($username)||empty($psw)){    //用户名或密码为空则提示重新填写
        echo '<h1>警告</h1>';
        echo '用户名或密码都不可以为空，请<a href=register.html>重新填写</a>';
    }elseif($arr!=NULL)                  //已经有存在的用户名则提示重新填写
        echo '该用户名已经被抢注，请<a href=register.html>重新填写</a>';
    elseif(!empty($username)&&!empty($psw)){
                                 //用户名和密码不为空且不重复则将数据写入数据库
        $sql="INSERT phppan(username,password) VALUES('{$username}',
                            '{$psw}')";
        mysql_query($sql,$link)or die('数据库出现问题！'
                                      .mysql_error($link));
        echo "账号注册成功！现在你可以去<a href=login.html>登录</a>";
        mkdir("../{$username}"); //创建于用户名同名的文件夹作为网盘主文件夹
    }
?>
```

该段 PHP 代码重点就在于判断注册页面传递的用户名和密码是否为空，或者已经有存

在的用户名。成功注册后则为该用户创建一个同用户名相同的主文件夹。

（2）登录实现

登录的实现同注册类似，HTML 页面实现代码如下：

```
<!--login.html-->
<html>
    <h1>登录网盘</h1>
    <form action="login.php" method="post">
        用户名：
        <input type="text" name="login_name"><br /><br />
        密    码：
        <input type="text" name="login_password"><br />
        <input type="submit" value="登录">
    </form>
</html>
```

PHP 代码实现如下：

```
/*login.php*/
<?php
    error_reporting(NULL);                      //关闭错误提示
    session_start();                            //开启 session
    setcookie(session_name(),session_id());//设置 cookie
    $link=mysql_connect('localhost','root','');
    mysql_select_db('mydatabase',$link);
    $username=$_POST['login_name'];            //获取 HTML 传递的用户名
    $sql="SELECT password FROM phppan WHERE username='{$username}'";
    $res=mysql_query($sql,$link);
    $arr=mysql_fetch_assoc($res);              //获取对应用户名的密码
    if($arr['password']!==$_POST['login_password']){
                                                //判断用户名与密码是否匹配
        echo "<h1>登录失败</h1>";
        echo "用户名或者密码错误，请重新<a href=login.html>登录</a>";
    }else{                                      //登录成功则设置 session 变量
        $_SESSION['username']=$username;       //用来记录用户名
        $_SESSION['dir']=$username;            //用来记录用户目录
        echo '<h1>登录成功</h1>';
        echo '马上跳转至管理页面...';
        echo "<meta http-equiv=\"refresh\" content=\"1;url=
                managefile.php\">";             //延迟跳转
    }
?>
```

该段代码的重点在于成功登录后设置记录用户名和用户目录的 session 变量。

3. 管理文件

文件管理是网盘程序的核心，其实现代码如下：

```
/*managefile.php*/
<?php
    error_reporting(NULL);
    session_start();                        //取回登录用户名
    if(isset($_GET['back'])&&($_GET['back']=='yes'))
                                            //判断是否单击返回上一级链接
        $_SESSION['dir']=substr($_SESSION['dir'],0, stripos
```

```
                ($_SESSION['dir'],'/'));    //将 session 中存储的路径减少一级
    if(isset($_GET['dir'])){                //判断是否单击目录链接
        $_SESSION['dir'].="/{$_GET['dir']}";
                                   //进入子目录则为 session 变量中路径加入子目录
        chdir("../{$_SESSION['dir']}");
                                   //改变目录以供 scandir 遍历目录文件
    }else
        chdir("../{$_SESSION['dir']}");//没有进入子目录则展示主目录文件
    if($_SESSION['dir']!=$_SESSION['username'])
                                            //判断是否需要显示返回上一级链接
        echo "<a href=managefile.php?back=yes>返回上一级</a><br />";
    $arr=scandir(getcwd());                 //获取目录中的文件并返回
    echo "<table>";                         //将结果装入 HTML 表格
    foreach($arr as $v){                    //遍历返回的结果集
        if(is_dir("./{$v}")&&$v!='.'&&$v!='..')
                                            //如果文件为目录则显示为链接
            echo "<tr><a href=managefile.php?dir={$v}>{$v}</a></tr>";
        elseif(is_file($v))                 //如果文件为普通文件则提供下载链接
            echo "<tr><td>{$v}</td><td><a href=down.php?filename=
                                            {$v}>下载</tr>";
    }
    echo "</table>";
?>
```

该文件的难点在于进入子目录和返回上一级目录，这就需要对当前 session 变量了解很透彻，然后对该变量进行增加目录（进入子目录）和减少目录（返回上级目录）。每个目录都应该提供上传文件的选项，因此该部分可以写入到 managefile.php 文件中：

```
/*managefile.php*/
<form enctype='multipart/form-data' method='post'>
    <input type='file' name='usrfile' />
    <input type='submit' value='上传'>
</form>
<?php
    if(!empty($_FILES)){                                  //判断变量是否为空
        $filename=$_FILES['usrfile']['tmp_name'];         //定义要转移的文件名
        $filedir="../{$_SESSION['dir']}/{$_FILES['usrfile']['name']}";
                                                          //定义目标路径
        move_uploaded_file($filename,$filedir);
        echo "<meta http-equiv=\"refresh\" content=\"0;url=
                managefile.php\">";                       //立即刷新页面
    } else{
    }
?>
```

该段代码中的重点在于为 move_uploaded_file 提供正确的目标路径，同时在上传后立即刷新该页面，以便用户管理新上传的文件。

4. 文件下载

在文件管理页面中点击对应文件名后的下载链接即可下载对应的文件，需要做的就是改变报头并提供文件名称以供浏览器下载，实现的代码如下：

```
/*down.php*/
<?php
    error_reporting(NULL);
    //发送指定的报头
    Header("Content-type: application/octet-stream");
    Header("Content-Disposition: attachment; filename="
```

```
                    .$_GET['filename']);
06  ?>
```

14.1.3　简易网盘运行测试

在编码过程中通常需要非常多的测试过程，这里我们就来测试一下程序是否以我们期望的效果运行。

1. 注册测试

注册测试通常是测试在一些可能错误的情况下程序能否做出正确的响应。例如我们不填写用户名和密码就进行提交，程序就会出现如图 14.7 所示的响应。可以看到，程序并不会让这种情况成功注册。我们在填写用户名和密码后单击“注册”按钮，应该出现如图 14.8 所示的响应。而我们以同样的用户名再次注册的时候，程序应该阻止该用户注册，会出现如图 14.9 所示的响应。



图 14.7　警告信息



图 14.8　成功注册



图 14.9　阻止同名用户注册

经过以上 3 种特殊情况的测试，我们可以看到程序都做出了期望的响应。

2. 登录测试

登录测试非常简单，只需要测试用户名对应的密码是否与数据库中的数据一致，如果不一致则提示用户重新输入，如图 14.10 所示。如果提交的用户名和密码是对应的，则跳转到管理页面，如图 14.11 所示。



图 14.10　用户名和密码不一致



图 14.11　跳转到管理页面

3. 文件浏览测试

在成功登录后浏览器会跳转到管理文件的页面，我们需要测试的就是能否进入子文件夹，并且可以从子文件夹中返回上一级文件夹。单击管理页面中的文件夹连接，应该出现如图 14.12 所示的界面。当单击返回上一级连接后，应该返回上一级菜单，如图 14.13 所示。



图 14.12　进入子文件夹



图 14.13　从子文件中返回

可以看到，程序已经显示子文件夹上一级目录中的文件。

4. 文件上传测试

文件上传测试则主要测试上传后的文件能否立即被显示，我们选择一个文件上传，应该会出现图 14.14 所示的响应。在上传文件后并不需要手动刷新就可以显示刚刚上传的文件，这是我们所期望的。

5. 文件下载测试

文件下载测试非常简单，只需单击对应文件名之后的下载链接来查看浏览器是否会出现下载选择，如图 14.15 所示。



图 14.14　上传文件



图 14.15　下载文件测试

从浏览器的正确响应我们可以得知，文件的下载功能已经被正确实现。到此为止，网盘的全部功能已基本成功实现。

14.2　简易投票系统功能设计

在现在的 Web 站点中或者论坛中，常常可以见到一些投票统计的模块。本节我们就来使用 PHP 实现一个简易的投票系统。

14.2.1　功能设计

与网盘设计类似，这里我们也来构思一下一个投票系统需要实现的功能。

1. 简易管理

该功能是用来管理投票的主题，例如可以添加一个新的投票主题，也可以查看已经存在的投票主题。该页面可以非常简单，只需要两个跳转至相关处理页面的链接即可，如图 14.16 所示。

2. 显示存在的主题

显示存在的主题是管理项目的一项基本功能，它可以转到显示数据库中已经存在的投票主题，如图 14.17 所示。每个存在的主题后应该提供浏览链接，该链接可以查看该主题并且可以模拟投票，如图 14.18 所示。



图 14.16　管理界面



图 14.17　显示存在的主题



图 14.18　浏览存在的主题

如果当前数据库中并没有存在的主题，则显示提示信息，如图 14.19 所示。单击提示信息提供的“添加一个主题”链接可以跳转到添加主题页面。

3. 添加投票主题

在主题管理页和数据库中不存在投票主题的时候，我们就可以单击添加主题链接以进入添加主题页面，如图 14.20 所示。



图 14.19　没有存在的主题时的提示信息



图 14.20　添加投票主题

在该页面中，可以指定投票主题的名称、投票的方式和选项的个数。该选项会将用户填入的信息保存在数据库，并且转向指定投票选项的页面，需要注意的是，指定投票选项的表格应该是动态生成的，可以随着选项的个数不同而不同，例如选项个数为 2，则应该如图 14.21 所示；如果选项个数为 5，则应该如图 14.22 所示。

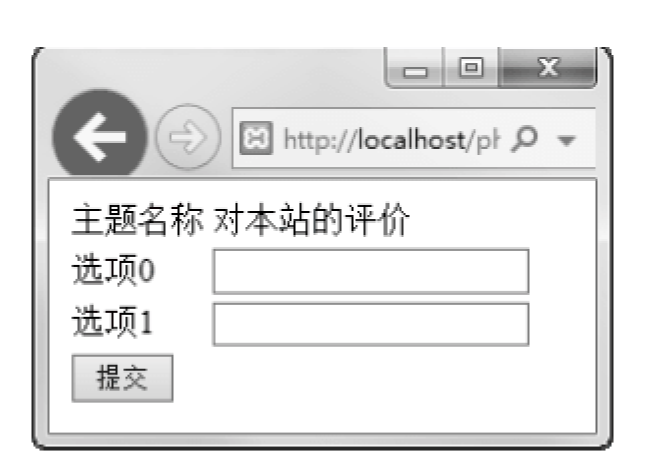


图 14.21　添加投票主题



图 14.22　添加投票主题

在该页面提交的信息会写入数据库中。

4. 模拟投票

模拟投票就是通过浏览已经存在的投票主题，然后进行投票，投票数据会被保存在数据库中。我们选择一项并进行投票，如图 14.23 所示。在投票提交后，出现如图 14.24 所示的页面。

我们单击提供的“查看投票结果”链接，可以查看投票的结果，如图 14.25 所示。



图 14.23　进行投票



图 14.24　投票成功



图 14.25　投票结果

从投票结果可以看到，“满意”选项已经被投票一次。

5. 数据库设计

在投票系统中，可以固定设计的数据库只有一个简单的用来记录投票主题的数据库。SQL 语句如下：

```
CREATE TABLE theme_name (
    id TINYINT AUTO_INCREMENT,
    name VARCHAR(100),
    mode VARCHAR(10)
);
```

创建后的结构表数据如下：

```
mysql> DESCRIBE theme_name;
+-------+--------------+-------+-------+----------+------------------+
| Field | Type         | Null  | Key   | Default  | Extra            |
+-------+--------------+-------+-------+----------+------------------+
| id    | tinyint(4)   | NO    | PRI   | NULL     | auto_increment   |
| name  | varchar(100) | NO    |       | NULL     |                  |
| mode  | varchar(10)  | NO    |       | NULL     |                  |
+-------+--------------+-------+-------+----------+------------------+
3 rows in set (0.01 sec)
```

14.2.2　具体代码实现

本节就以代码的形式实现 14.2.1 节中的功能设计。

1. 管理页面

管理页面实现非常简单，只需要一个简单的 HTML 页面即可：

```
<!--index.html-->
<html>
    <h1>投票主题管理</h1>
    <a href=show_theme.php>显示存在的主题</a><br />
    <a href=add.html>添加主题</a>
</html>
```

该页面中的两个链接用来跳转至相关操作的页面。

2. 显示存在的主题

显示存在的主题页面实现也是比较简单的，该页面通过读取数据库来判断是否存在投票主题。如果存在投票主题则展示这些主题，如不存在则提示相关信息：

```
/*show_theme.php*/
<?php
    header("Content-Type:text/html;charset=utf-8");    //发送自定义报头
    $link=mysql_connect('localhost','root','');        //连接数据库
    mysql_select_db('mydatabase',$link);
    mysql_query("SET NAMES 'utf8'");               //设置数据库相关编码
    mysql_query("SET CHARACTER_SET_CLIENT=utf8");
    mysql_query("SET CHARACTER_SET_RESULTS=utf8");
    $sql="SELECT * FROM theme_name";
    $res=mysql_query($sql,$link)or die(mysql_error()); //查询投票主题
    if(!mysql_affected_rows($link)){    //不存在投票主题则发出提示信息
        echo "<h1>警告！</h1>";
        echo '当前还没有主题可以显示，您可以';
        echo '<a href=add.html>添加一个主题</a>';
    }else{                              //存在投票主题则展示存在的投票主题
        echo "<h1>当前存在的主题</h1><table>";
        while($arr=mysql_fetch_assoc($res)){
            echo "<tr><td>{$arr['id']}</td><td>{$arr['name']}</td>
                <td>
            <a href=ballot.php?id={$arr['id']}&mode={$arr['mode']}>
                浏览</a></td></tr>";
        }
        echo "</table>";
    }
?>
```

3. 添加投票主题

添加投票主题是投票系统的难点，因为实现代码会从一个数据表中查询相关信息并作为创建新数据表的信息，其中的数据使用是比较复杂的。

```
/*add.php*/
```

```
<?php
    session_start();                                        //开启 Session
    setcookie(session_name(),session_id());                 //设置 Cookie
    header("Content-Type:text/html;charset=utf-8");
    if(!empty($_POST)){                                     //设置 Session 变量
        $_SESSION['theme_name']=$_POST['theme_name'];
        $_SESSION['mode']=$_POST['mode'];
        $_SESSION['amount']=$_POST['amount'];
    }
?>
<form method='post' action='add_db.php'>
<table>
    <tr><td>主题名称</td><td><?php echo $_SESSION['theme_name'];?></td>
        </tr>
<?php
    for($i=0;$i<$_SESSION['amount'];$i++){                  //动态输出表格
        echo "<tr><td>选项{$i}</td><td><input type='text' name=
            '{$i}'/></td></tr>";
    }
?>
<tr><td><input type='submit' value='提交'></td></tr>
</table></form>
```

用户在表格中添加相应的数据后，该页面的信息会传入 add_db.php 文件：

```
/*add_db.php*/
<?php
    session_start();                                        //取回 Session
    header("Content-Type:text/html;charset=utf-8");
    $link=mysql_connect('localhost','root','');
    mysql_select_db('mydatabase',$link);
    mysql_query("SET NAMES 'utf8'");                        //设置数据库编码
    mysql_query("SET CHARACTER_SET_CLIENT=utf8");
    mysql_query("SET CHARACTER_SET_RESULTS=utf8");
    //为 theme_name 表添加数据
    $sql="INSERT theme_name (name,mode) ";
    $sql.= "VALUES('{$_SESSION['theme_name']}',
               '{$_SESSION['mode']}')";
    mysql_query($sql,$link);
    //创建对应 Id 的数据表
    $sql="SELECT id from theme_name WHERE name=
            '{$_SESSION['theme_name']}'";
    $res=mysql_query($sql,$link)or die(mysql_error());
    $arr=mysql_fetch_assoc($res);
    $table_id=$arr['id'];
    $sql="CREATE TABLE IF NOT EXISTS ";
    $sql.="'{$table_id}'(id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,";
    $sql.="options VARCHAR(100) NOT NULL,number INT DEFAULT 0)";
    mysql_query($sql,$link)or die(mysql_error());
    $options='';
    for($i=0;$i<$_SESSION['amount'];$i++)
        $options.="('{$_POST[$i]}')".',';
                                  //将投票主题的选项连接作为 SQL 语句
    $options=substr($options,0,strlen($options)-1);
                                  //去掉循环生成的 SQL 语句后的多余逗号
    $sql="INSERT '{$table_id}' (options) VALUES {$options}";
    mysql_query($sql,$link)or die(mysql_error());
                                  //向新生成的表中添加数据
?>
```

```
<h1>添加主题成功！</h1>
您可以<a href=show_theme.php>查看</a>存在的主题
```

4. 模拟投票

模拟投票在投票系统中用来更新选项对应的票数，实现也是简单的：

```
/*ballot.php*/
<?php
    header("Content-Type:text/html;charset=utf-8");
    $id=$_GET['id'];                            //获取投票主题的 ID
    $mode=$_GET['mode'];                        //获取投票主题的投票方式
    $link=mysql_connect('localhost','root','');
    mysql_select_db('mydatabase',$link);
    mysql_query("SET NAMES 'utf8'");
    mysql_query("SET CHARACTER_SET_CLIENT=utf8");
    mysql_query("SET CHARACTER_SET_RESULTS=utf8");
    $sql="SELECT name FROM theme_name WHERE id={$id}";
                                                //将查询结果作为投票主题名
    $res=mysql_query($sql,$link)or die(mysql_error());
    $arr=mysql_fetch_assoc($res);
    echo "<form action=update.php?id={$id}&mode={$mode} method=post>
            <table>";
    echo "<tr><td></td><th>{$arr['name']}</th></tr>";
    $sql="SELECT options FROM '{$id}'";      //查询对应表中的所有投票选项
    $res=mysql_query($sql,$link)or die(mysql_error());
    $i=0;                                       //作为标识每个选项的 ID
    while($arr=mysql_fetch_assoc($res)){//根据投票的模式生成对应的投票列表
        if($mode=='radio'){
            $i++;
            echo "<tr><td><input type=radio name=vote value={$i}>
                  </td>";
            echo "<td>{$arr['options']}</td></tr>";
        }else{
            $i++;
            echo "<tr><td><input type=checkbox name=id[] value={$i}>
                  </td>";
            echo "<td>{$arr['options']}</td></tr>";
        }
    }
    echo "<tr><td></td><td><input type=submit value=投票 /></td></tr>
           </table></form>";
    echo "<br />您还可以返回<a href=show_theme.php>继续浏览</a>";
?>
```

5. 投票统计

投票统计就是用来查询对应的投票数，只需要通过查询对应数据表的投票选项名称和数目，然后循环列出即可。

```
/*show_res.php*/
<?php
    header("Content-Type:text/html;charset=utf-8");
    $id=$_GET['id'];                            //获取要查看的表的 ID
    $link=mysql_connect('localhost','root','');
    mysql_select_db('mydatabase',$link);
    mysql_query("SET NAMES 'utf8'");
    mysql_query("SET CHARACTER_SET_CLIENT=utf8");
    mysql_query("SET CHARACTER_SET_RESULTS=utf8");
```

```
    $sql="SELECT options,number FROM '{$id}'";
                                        //获取对应的投票选项及票数
    $res=mysql_query($sql,$link)or die(mysql_error());
    echo '<table border=1>';
    while($arr=mysql_fetch_assoc($res)){    //将查询结果输出到表格
        echo "<tr><td>{$arr['options']}</td><td>{$arr['number']}
              </td></tr>";
    }
    echo '</table><br />';
    echo '<a href=index.php>返回管理页</a>';
?>
```

代码实现部分的难点就在于，PHP 代码需要对应保存投票主题的表 theme_name，去创建对应保存投票选项的数据表。如果读者在这一部分有非常清晰的思路，则实现该投票系统是会非常轻松的。

14.2.3　运行测试

在投票系统中，没有类似网盘的登录程序及对应的目录操作的程序，更多的是数据库操作的知识。因此我们也没有必要进行太多的例外测试，我们就按常规正确的操作来测试这一系统能否正确工作。

1. 添加和查看投票主题

添加投票主题可以从管理页面进入，首先出现的页面是对将要添加的投票主题做设置。例如，设置主题的名称、投票模式（单选或者复选）和选项的个数。这里我们就测试分别添加一个单选投票主题和一个复选投票主题，如图 14.26 所示。在添加成功后就可以查看当前存在的主题，如图 14.27 所示。



图 14.26　添加投票主题

通过浏览相应的主题会出现投票主题的结构及模拟投票的选项，如图 14.28 所示。



图 14.27　当前存在的主题　　　　图 14.28　浏览投票主题

单选投票与复选投票主题的不同点就在于选择框的不同。

2. 模拟投票和查看投票结果

在浏览投票主题的同时我们可以进行模拟投票，如图 14.29 所示。

在投票成功后就可以查看投票的结果，如图 14.30 所示。

对应图 14.29 模拟投票的选项可以看到投票结果都进行了正确的记录。这就证明我们的投票系统是可以正常使用的。

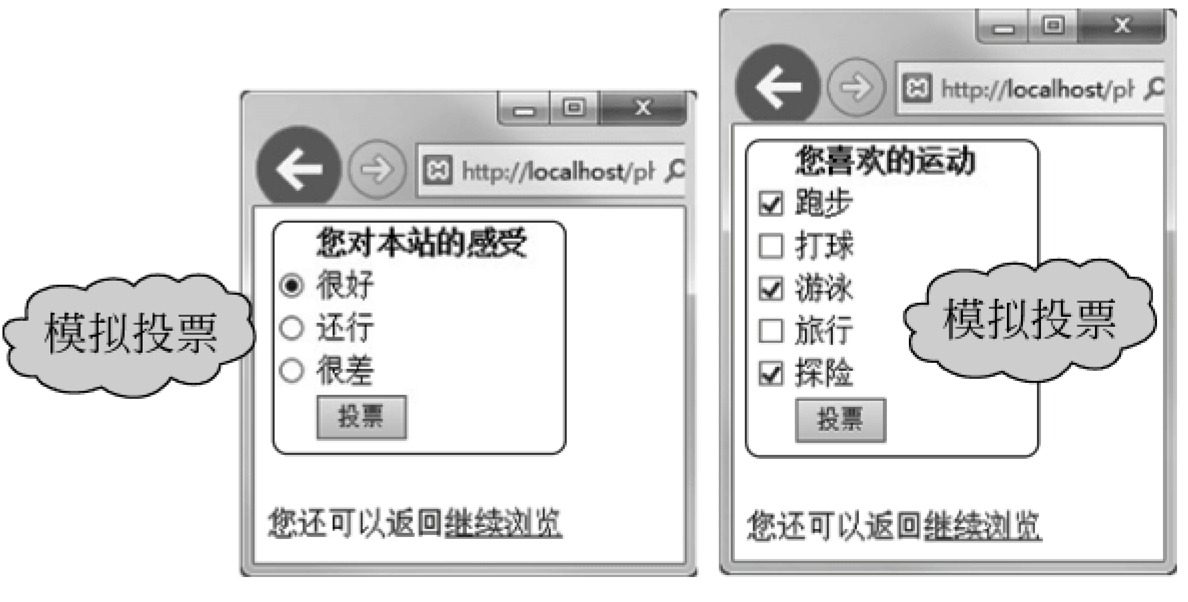



图 14.29　模拟投票



图 14.30　投票结果

14.3　小　　结

本章主要使用 PHP 编写实现了简易网盘和投票系统，其中使用到了字符串处理、文件系统、数据库系统及多个全局变量等非常多的知识。但是这两个模块只是实现了最基本的功能，远远不能称得上完整，例如，还可以为网盘添加创建文件夹、获取文件的详细信息等功能；为投票系统添加修改投票参数等功能，这就需要读者自己去尝试了。